200+ JavaScript Programs for Beginners

JS

by Hernando Abella

200+ JavaScript Programs for Beginners

By Hernando Abella

ALUNA PUBLISHING HOUSE

Thank you for trusting our Publishing House. If you could evaluate our work and give us a comment on Amazon, we will appreciate it very much!

This Book may not be copied or printed without the permission of the author.

COPYRIGHT 2024 ALUNA PUBLISHING HOUSE

Table of contents

Intro

1. Print Hello World

2. Add Two Numbers

3. Find the Square Root

4. Calculate the Area of a Triangle

5. Swap Two Variables

6. Solve Quadratic Equation

7. Convert Kilometers to Miles

8. Convert Celsius to Fahrenheit

9. Generate a Random Number

10. Check if a number is Positive, Negative, or Zero

11. Check if a Number is Odd or Even

12. Find the Largest Among Three Numbers

13. Check Prime Number

14. Print All Prime Numbers in an Interval

15. Find the Factorial of a Number

16. Display the Multiplication Table

17. Print the Fibonacci Sequence

18. Check Armstrong Number

19. Find Armstrong Number in an Interval

20. Make a Simple Calculator

21. Find the Sum of Natural Numbers

22. Check if the Numbers Have the Same Last Digit

23. Find HCF or GCD

24. Find LCM

25. Find the Factors of a Number

26. Find Sum of Natural Numbers Using Recursion

27. Guess a Random Number

28. Shuffle Deck of Cards

29. Display Fibonacci Sequence Using Recursion

30. Find Factorial of Number Using Recursion

31. Convert Decimal to Binary

32. Find ASCII Value of Character

33. Check Whether a String is Palindrome or Not

34. Sort Words in Alphabetical Order

35. Replace Characters of a String

36. Reverse a String

37. Check the Number of Occurrences of a Character in the String

38. Convert the First Letter of a String into UpperCase

39. Count the Number of Vowels in a String

40. Check Whether a String Starts and Ends With Certain Characters

41. Replace All Occurrences of a String

42. Create Multiline Strings

43. Format Numbers as Currency Strings

44. Generate Random String

45. Check if a String Starts with Another String

46. Trim a String

47. Check Whether a String Contains a Substring

48. Compare Two Strings

49. Encode a String to Base64

50. Replace all Instances of a Character in a String

51. Replace All Line Breaks with

52. Check Leap Year

53. Format the Date

54. Display Current Date

55. Compare The Value of Two Dates

56. Create Countdown Timer

57. Remove Specific Item From an Array

58. Check if An Array Contains a Specified Value

59. Insert Item in an Array

60. Get Random Item From an Array

61. Perform Intersection Between Two Arrays

62. Split Array into Smaller Chunks

63. Get File Extension

64. Check If a Variable Is undefined or null

65. Generate a Random Number Between Two Numbers

66. Get The Current URL

67. Validate An Email Address

68. Check If a Variable is of Function Type

69. Work With Constants

70. Pass Parameter to a setTimeout() Function

71. Generate a Range of Numbers and Characters

72. Perform Function Overloading

73. Implement a Stack

74. Implement a Queue

75. Check if a Number is Float or Integer

76. Pass a Function as Parameter

77. Get the Dimensions of an Image

78. Remove All Whitespaces From a Text

79. Write to Console

80. Convert Date to Number

81. Find the Average of Two Numbers

82. Calculate the Area of a Circle

83. Random Color Generator

84. Check if a String is Empty

85. Capitalize the First Letter of a String

86. Find the Maximum Element in an Array

87. Reverse an Array

88. Calculate the Power of a Number

89. Find the Minimum Element in an Array

90. Convert Minutes to Hours and Minutes

91. Find the Sum of Digits in a Number

92. Check if a String is a Palindromic Phrase

93. Generate a Random Password

94. Calculate Simple Interest

95. Implement a Basic Stopwatch

96. Check if a Number is a Perfect Number

97. This program calculates the Volume of a Cylinder

98. Generate a Random Quote

99. Find the Intersection of Two Arrays

100. Convert Feet to Meters

101. Convert Days to Years, Months, and Days

102. Find the Median of an Array

103. Calculate the Distance Between Two Points

104. Check if a Number is a Perfect Square

105. Find the Area of a Rectangle

106. Convert Binary to Decimal

107. Count the Number of Words in a Sentence

108. Find the Union of Two Arrays

109. Calculate the Factorial of a Large Number

110. Check if a Number is a Strong Number

111. Check if a Number is a Narcissistic Number

112. Count the Number of Consonants in a String

113. Check if a Number is a Triangular Number

114. Find the Area of a Trapezoid

115. Calculate the Greatest Common Divisor (GCD) of Two Numbers

116. Check if a Number is a Fibonacci Number

117. Find the Perimeter of a Rectangle

118. Convert Roman Numerals to Numbers

119. Check if a String is Anagram of Another String

120. Generate Pascal's Triangle

121. Convert Decimal to Roman Numerals

122. Find the Area of a Parallelogram

123. Basic Rock, Paper, Scissors Game

124. Basic Slot Machine Game

125. Check if a Number is a Smith Number

126. Basic Chessboard

127. Generate a Random Password with Specific Criteria

128. Basic Maze Solver

129. Check if a String is Pangram or Not

130. Basic Sudoku Solver

131. Convert Objects to Strings

132. Find the Area of an Ellipse

133. Check if a Number is a Palindrome in Binary

134. Find the Area of a Rhombus

135. Check if a Number is a Catalan Number

136. Find the Luhn Algorithm Check Digit

137. Basic Maze Generator

138. Check if a Year is a Magic Year

139. Basic Crossword Puzzle

140. Find the Area of a Regular Polygon

141. Check if a Number is an Abundant Number

142. Basic Word Guessing Game

143. Calculate the Euler's Totient Function

144. Basic Memory Matching Game

145. Check if a Number is a Vampire Number

146. Basic Simon Says Game

147. Calculate the Area of a Hexagon

148. Check if a Number is a Pronic Number

149. Basic Minesweeper Game

150. Find the Area of a Pentagon

151. Check if a Number is a Cube Number

152. Basic Typing Speed Test

153. Find the Area of a Cube

154. Find the Area of a Cone

155. Check if a Number is a Happy Number

156. Calculate the Area of a Triangular Prism

157. Check if a Number is a Fibonacci Prime

158. Check if a Number is a Squareful Number

159. Find the Area of a Tetrahedron

160. Check if a Number is an Automorphic Number

161. Calculate the Area of a Pyramid

162. Check if a Number is a Smith–Morra Gambit Number

163. Check if a Number is a Solitary Number

164. Basic Tower of Hanoi Puzzle

165. Calculate the Area of a Frustum

166. Check if a Number is a Motzkin Number

167. Check if a Number is a Padovan Number

168. Calculate the Area of a Koch Snowflake

169. Basic Chatbot

170. RGB to HEX Converter

171. Counter

172. Stop The Bomb

173. IP Address Generator

174. Digital clock

175. Random Number Generator

176. Simple JavaScript Quiz

177. Text-based RPG Adventure

178. Dice Rolling Simulator

179. Seconds to Time Converter

180. Bar Chart Generator

181. Right-Angled Triangle Pattern

182. Text-Based Calculator

183. Number Pyramid Generator

184. Diamond Pattern Generator

185. Hangman Game

186. Countdown Timer

187. Mad Libs Game

188. Guess the Number Game

189. Whack-a-Mole Game

190. Coin Tossing Game

191. Fibonacci Number Generator

192. Math Quiz Game

193. Color Guessing Game

194. Word Scramble Game

195. Country Capital Quiz Game

196. Guess the Animal

197. Joke Teller

198. Shopping List

199. FizzBuzz Game

200. Swap Pairs of Adjacent Digits

201. Capitalization Changer

202. Array Halves Swapper

203. Sum of Digits in String

204. Sum of Cubes

205. Maximum Integer for Sum

206. URL Breakdown

207. Sort Strings by Length

208. Simplify Absolute Path

209. Count Common Elements in Arrays

210. Check Same Digits in a Number

211. Rightmost Round Number Position

212. Reverse Bits of 16-Bit Unsigned Short Integer

213. Greater Than 15 Checker

214. Replace First Digit with $

215. Prefix Sums

216. Next Prime Number

217. Reverse Order of Bits

218. Pyramid

Intro

Programming in JS was complicated until you appreciated this wonderful book.

Focus on the most important thing writing code, starting from the first program to the last will help you quickly increase your skills as a programmer and will give you the knowledge necessary to understand:

The flow of a program, the syntax, the direction, how it works, what Add keywords to your code, make great decisions, and much more.

This is a very essential guide with which you can learn to create more than... Once you have solved a certain number of programs, you will be able to develop yourself and turn your own ideas into a reality without even having to think about it.

1. Print Hello World

The "Hello World" program is a classic in programming and is often the first program written when learning a new programming language. In this case, the goal is to print the text string "Hello, World!" to the browser console.

```
console.log("Hello, World!");
```

2. Add Two Numbers

This program focuses on taking two numbers as input and then displaying their sum. It prompts the user to enter two numbers, adds them together, and prints the result.

```
let num1 = parseFloat(prompt("Enter the first number:"));
let num2 = parseFloat(prompt("Enter the second number:"));

if (!isNaN(num1) && !isNaN(num2)) {
  let sum = num1 + num2;
  console.log(`The sum of ${num1} and ${num2} is: ${sum}`);
} else {
  console.log("Please enter valid numbers.");
}
```

3. Find the Square Root

This program prompts the user to enter a number and calculates its square root. It then displays the result, ensuring that the entered number is non-negative.

```
let inputNumber = parseFloat(prompt("Enter a non-negative number:"));

if (!isNaN(inputNumber) && inputNumber >= 0) {
  let squareRoot = Math.sqrt(inputNumber);
  console.log(`The square root of ${inputNumber} is: ${squareRoot}`);
} else {
  console.log("Please enter a valid non-negative number.");
}
```

4. Calculate the Area of a Triangle

This program prompts the user to enter the base and height of a triangle and calculates its area using the formula: $Area = 12 \times Base \times Height$ $Area = 21 \times Base \times Height$. It then displays the calculated area.

```
let base = parseFloat(prompt("Enter the base of the triangle:"));
let height = parseFloat(prompt("Enter the height of the triangle:"));

if (!isNaN(base) && !isNaN(height) && base > 0 && height > 0) {
  let area = 0.5 * base * height;
  console.log(
    `The area of the triangle with base ${base} and height ${height} is: ${area}`
  );
} else {
  console.log("Please enter valid positive numbers for base and height.");
}
```

5. Swap Two Variables

This program prompts the user to enter two variables and swaps their values. It then displays the variables before and after the swap.

```
let variable1 = prompt("Enter the first variable:");
let variable2 = prompt("Enter the second variable:");

console.log(
  `Before swapping: Variable1 = ${variable1}, Variable2 = ${variable2}`
);

// Swapping the variables
let temp = variable1;
variable1 = variable2;
variable2 = temp;

console.log(
  `After swapping: Variable1 = ${variable1}, Variable2 = ${variable2}`
);
```

6. Solve Quadratic Equation

This program solves a quadratic equation of the form ax^2 + bx + c = 0, where a, b, and c are coefficients. It uses the quadratic formula to find the roots of the equation.

```
// Prompt user for coefficients
let a = parseFloat(prompt("Enter the coefficient a:"));
let b = parseFloat(prompt("Enter the coefficient b:"));
let c = parseFloat(prompt("Enter the coefficient c:"));

// Calculate the discriminant
let discriminant = b ** 2 - 4 * a * c;

// Check if roots are real
if (!isNaN(a) && !isNaN(b) && !isNaN(c)) {
 if (discriminant > 0) {
  let root1 = (-b + Math.sqrt(discriminant)) / (2 * a);
  let root2 = (-b - Math.sqrt(discriminant)) / (2 * a);
  console.log(
   `The roots of the quadratic equation are: ${root1} and ${root2}`
  );
 } else if (discriminant === 0) {
  let root = -b / (2 * a);
  console.log(`The quadratic equation has a repeated root: ${root}`);
 } else {
```

```
    console.log("The quadratic equation has complex roots.");
  }
} else {
  console.log("Please enter valid numbers for coefficients.");
}
```

7. Convert Kilometers to Miles

This program prompts the user to enter a distance in kilometers and converts it to miles using the conversion factor: 1 kilometer is approximately equal to 0.621371 miles. It then displays the converted distance.

```
// Prompt user for distance in kilometers
let kilometers = parseFloat(prompt("Enter the distance in kilometers:"));

// Conversion factor
const kilometersToMilesConversionFactor = 0.621371;

// Check if input is a valid number
if (!isNaN(kilometers)) {
  // Convert kilometers to miles
  let miles = kilometers * kilometersToMilesConversionFactor;
  console.log(`${kilometers} kilometers is approximately ${miles} miles.`);
} else {
  console.log("Please enter a valid number for the distance in kilometers.");
}
```

8. Convert Celsius to Fahrenheit

This program converts a temperature from Celsius to Fahrenheit using the formula: **F = (C * 9/5) + 32**, where F is the temperature in Fahrenheit and C is the temperature in Celsius.

```
// Prompt user for temperature in Celsius
let celsius = parseFloat(prompt("Enter the temperature in Celsius:"));

// Check if input is a valid number
if (!isNaN(celsius)) {
  // Convert Celsius to Fahrenheit
  let fahrenheit = (celsius * 9) / 5 + 32;
  console.log(
    `${celsius} degrees Celsius is equal to ${fahrenheit} degrees Fahrenheit.`
  );
} else {
  console.log("Please enter a valid number for the temperature in Celsius.");
}
```

9. Generate a Random Number

This program generates a random number between a user-defined range. It prompts the user to enter the minimum and maximum values of the range, and then it displays the generated random number within that range.

```
// Prompt user for the range
let minRange = parseFloat(prompt("Enter the minimum value of the range:"));
let maxRange = parseFloat(prompt("Enter the maximum value of the range:"));

// Check if input is a valid number
if (!isNaN(minRange) && !isNaN(maxRange) && minRange < maxRange) {
  // Generate a random number within the specified range
  let randomNumber = Math.random() * (maxRange - minRange) + minRange;
  console.log(
    `A random number between ${minRange} and ${maxRange} is: ${randomNumber}`
  );
} else {
  console.log(
    "Please enter valid numbers, ensuring that the minimum value is less than the maximum value."
  );
}
```

10. Check if a number is Positive, Negative, or Zero

This program prompts the user to enter a number and checks whether it is positive, negative, or zero. It then displays the result.

```
// Prompt user for a number
let number = parseFloat(prompt("Enter a number:"));

// Check if input is a valid number
if (!isNaN(number)) {
 // Check if the number is positive, negative, or zero
 if (number > 0) {
  console.log(`${number} is a positive number.`);
 } else if (number < 0) {
  console.log(`${number} is a negative number.`);
 } else {
  console.log("The entered number is zero.");
 }
} else {
 console.log("Please enter a valid number.");
}
```

11. Check if a Number is Odd or Even

This program prompts the user to enter a number and checks whether it is odd or even. It then displays the result.

```
// Prompt user for a number
let number = parseInt(prompt("Enter a number:"));

// Check if input is a valid integer
if (!isNaN(number) && Number.isInteger(number)) {
 // Check if the number is odd or even
 if (number % 2 === 0) {
  console.log(`${number} is an even number.`);
 } else {
  console.log(`${number} is an odd number.`);
 }
} else {
 console.log("Please enter a valid integer.");
}
```

12. Find the Largest Among Three Numbers

This program prompts the user to enter three numbers and determines and displays the largest among them.

```
// Prompt user for three numbers
let num1 = parseFloat(prompt("Enter the first number:"));
let num2 = parseFloat(prompt("Enter the second number:"));
let num3 = parseFloat(prompt("Enter the third number:"));

// Check if inputs are valid numbers
if (!isNaN(num1) && !isNaN(num2) && !isNaN(num3)) {
  // Find the largest among the three numbers
  let largestNumber = Math.max(num1, num2, num3);
  console.log(
    `The largest number among ${num1}, ${num2}, and ${num3} is: ${largestNumber}`
  );
} else {
  console.log("Please enter valid numbers for all three inputs.");
}
```

13. Check Prime Number

This program prompts the user to enter a number and checks whether it is a prime number or not. It then displays the result.

```
// Prompt user for a number
let number = parseInt(prompt("Enter a number:"));

// Check if input is a valid integer greater than 1
if (!isNaN(number) && Number.isInteger(number) && number > 1) {
 let isPrime = true;

 // Check for factors from 2 to the square root of the number
 for (let i = 2; i <= Math.sqrt(number); i++) {
  if (number % i === 0) {
   isPrime = false;
   break;
  }
 }

 // Display the result
 if (isPrime) {
  console.log(`${number} is a prime number.`);
 } else {
  console.log(`${number} is not a prime number.`);
```

```
  }
} else {
  console.log("Please enter a valid integer greater than 1.");
}
```

14. Print All Prime Numbers in an Interval

This program prompts the user to enter a starting and ending number for an interval and then prints all the prime numbers within that interval.

```
// Prompt user for the interval
let startNumber = parseInt(
  prompt("Enter the starting number of the interval:")
);
let endNumber = parseInt(prompt("Enter the ending number of the interval:"));

// Check if inputs are valid integers and the startNumber is less than the endNumber
if (
  !isNaN(startNumber) &&
  !isNaN(endNumber) &&
  Number.isInteger(startNumber) &&
  Number.isInteger(endNumber) &&
  startNumber < endNumber &&
  startNumber > 1
) {
  console.log(`Prime numbers in the interval [${startNumber}, ${endNumber}]:`);

  // Check for prime numbers in the interval
  for (let i = startNumber; i <= endNumber; i++) {
    let isPrime = true;
```

```
    for (let j = 2; j <= Math.sqrt(i); j++) {
      if (i % j === 0) {
        isPrime = false;
        break;
      }
    }

    if (isPrime) {
      console.log(i);
    }
  }
} else {
  console.log(
    "Please enter valid integers, ensuring that the starting number is less than the ending number and greater than 1."
  );
}
```

15. Find the Factorial of a Number

This program calculates the factorial of a given number. The factorial of a non-negative integer `n`, denoted as `n!`, is the product of all positive integers less than or equal to `n`.

```
// Prompt user for a non-negative integer
let number = parseInt(prompt("Enter a non-negative integer:"));

// Check if input is a valid non-negative integer
if (!isNaN(number) && Number.isInteger(number) && number >= 0) {
 // Calculate the factorial
 let factorial = 1;
 for (let i = 1; i <= number; i++) {
  factorial *= i;
 }

 console.log(`The factorial of ${number} is: ${factorial}`);
} else {
 console.log("Please enter a valid non-negative integer.");
}
```

16. Display the Multiplication Table

This program prompts the user to enter a number and displays its multiplication table up to a specified range (e.g., up to 10).

```
// Prompt user for a number
let number = parseInt(prompt("Enter a number for the multiplication table:"));

// Check if input is a valid integer
if (!isNaN(number) && Number.isInteger(number)) {
 // Specify the range for the multiplication table
 const range = 10;
 console.log(`Multiplication table for ${number} (up to ${range}):`);

 // Display the multiplication table
 for (let i = 1; i <= range; i++) {
  let result = number * i;
  console.log(`${number} x ${i} = ${result}`);
 }
} else {
 console.log("Please enter a valid integer.");
}
```

17. Print the Fibonacci Sequence

This program prompts the user to enter the number of terms and then prints the Fibonacci sequence up to that number of terms. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1.

```
// Prompt user for the number of terms in the Fibonacci sequence
let numTerms = parseInt(
  prompt("Enter the number of terms in the Fibonacci sequence:")
);

// Check if input is a valid integer
if (!isNaN(numTerms) && Number.isInteger(numTerms) && numTerms > 0) {
  console.log(`Fibonacci sequence of ${numTerms} terms:`);

  let fibArray = [0, 1];

  // Generate the Fibonacci sequence
  for (let i = 2; i < numTerms; i++) {
    fibArray[i] = fibArray[i - 1] + fibArray[i - 2];
  }

  console.log(fibArray.join(", "));
} else {
  console.log("Please enter a valid positive integer for the number of terms.");
}
```

18. Check Armstrong Number

This program prompts the user to enter a number and checks whether it is an Armstrong number or not. An Armstrong number (also known as a narcissistic number or pluperfect digital invariant) is a number that is the sum of its own digits each raised to the power of the number of digits in the number. For example, 153 is an Armstrong number because $13+53+33=153$ $1_3+5_3+3_3=153$.

```
// Prompt user for a number
let number = parseInt(prompt("Enter a number:"));

// Check if input is a valid positive integer
if (!isNaN(number) && Number.isInteger(number) && number > 0) {
 let originalNumber = number;
 let numberOfDigits = originalNumber.toString().length;
 let sum = 0;

 // Calculate the sum of digits each raised to the power of the number of digits
 while (originalNumber > 0) {
  let digit = originalNumber % 10;
  sum += Math.pow(digit, numberOfDigits);
  originalNumber = Math.floor(originalNumber / 10);
 }

 // Check if the number is an Armstrong number
```

```
  if (sum === number) {
    console.log(`${number} is an Armstrong number.`);
  } else {
    console.log(`${number} is not an Armstrong number.`);
  }
} else {
  console.log("Please enter a valid positive integer.");
}
```

19. Find Armstrong Number in an Interval

This program prompts the user to enter a range (start and end) and then prints all Armstrong numbers within that interval.

```
// Prompt user for the interval
let startNumber = parseInt(
  prompt("Enter the starting number of the interval:")
);
let endNumber = parseInt(prompt("Enter the ending number of the interval:"));

// Check if inputs are valid positive integers and the startNumber is less than the endNumber
if (
  !isNaN(startNumber) &&
  !isNaN(endNumber) &&
  Number.isInteger(startNumber) &&
  Number.isInteger(endNumber) &&
  startNumber > 0 &&
  startNumber < endNumber
) {
  console.log(
    `Armstrong numbers in the interval [${startNumber}, ${endNumber}]:`
  );

  // Check for Armstrong numbers in the interval
```

```
for (let i = startNumber; i <= endNumber; i++) {
 let originalNumber = i;
 let numberOfDigits = originalNumber.toString().length;
 let sum = 0;

 while (originalNumber > 0) {
  let digit = originalNumber % 10;
  sum += Math.pow(digit, numberOfDigits);
  originalNumber = Math.floor(originalNumber / 10);
 }

 if (sum === i) {
  console.log(i);
  }
 }
} else {
 console.log(
  "Please enter valid positive integers, ensuring that the starting number is less than the ending number and both are greater than 0."
 );
}
```

20. Make a Simple Calculator

This program implements a simple calculator that can perform basic arithmetic operations. It prompts the user to enter two numbers and choose an operation (addition, subtraction, multiplication, or division). It then performs the selected operation and displays the result.

```
// Prompt user for two numbers
let num1 = parseFloat(prompt("Enter the first number:"));
let num2 = parseFloat(prompt("Enter the second number:"));

// Check if inputs are valid numbers
if (!isNaN(num1) && !isNaN(num2)) {
  // Prompt user for the operation
let operation = prompt(
  "Choose an operation (+ for addition, - for subtraction, * for multiplication, / for division):"
);

// Perform the selected operation
let result;
switch (operation) {
  case "+":
    result = num1 + num2;
    break;
  case "-":
    result = num1 - num2;
```

```
  break;
 case "*":
  result = num1 * num2;
  break;
 case "/":
  if (num2 !== 0) {
   result = num1 / num2;
  } else {
   console.log("Cannot divide by zero.");
   break;
  }
  break;
  default:
   console.log("Invalid operation.");
   break;
 }

 // Display the result
 if (result !== undefined) {
  console.log(`Result of ${num1} ${operation} ${num2} is: ${result}`);
 }
} else {
 console.log("Please enter valid numbers.");
}
```

21. Find the Sum of Natural Numbers

This program calculates the sum of natural numbers up to a given positive integer `n`. The sum is calculated using the formula: `sum = n * (n + 1) / 2`.

```
// Prompt user for a positive integer
let n = parseInt(prompt("Enter a positive integer:"));

// Check if input is a valid positive integer
if (!isNaN(n) && Number.isInteger(n) && n > 0) {
  // Calculate the sum of natural numbers
  let sum = (n * (n + 1)) / 2;
  console.log(`The sum of natural numbers from 1 to ${n} is: ${sum}`);
} else {
  console.log("Please enter a valid positive integer.");
}
```

22. Check if the Numbers Have the Same Last Digit

This program prompts the user to enter two numbers and checks if they have the same last digit.

```
// Prompt user for two numbers
let num1 = parseInt(prompt("Enter the first number:"));
let num2 = parseInt(prompt("Enter the second number:"));

// Check if inputs are valid integers
if (
  !isNaN(num1) &&
  !isNaN(num2) &&
  Number.isInteger(num1) &&
  Number.isInteger(num2)
) {
  // Extract the last digit of each number
  let lastDigit1 = Math.abs(num1 % 10);
  let lastDigit2 = Math.abs(num2 % 10);

  // Check if the last digits are the same
  if (lastDigit1 === lastDigit2) {
    console.log(
      `The last digit of ${num1} is the same as the last digit of ${num2}.`
    );
  } else {
```

```
    console.log(
      `The last digit of ${num1} is different from the last digit of ${num2}.`
    );
  }
} else {
  console.log("Please enter valid integers.");
}
```

23. Find HCF or GCD

This program prompts the user to enter two positive integers and calculates their Highest Common Factor (HCF) or Greatest Common Divisor (GCD).

```
// Prompt user for two positive integers
let num1 = parseInt(prompt("Enter the first positive integer:"));
let num2 = parseInt(prompt("Enter the second positive integer:"));

// Check if inputs are valid positive integers
if (!isNaN(num1) && !isNaN(num2) && Number.isInteger(num1) && Number.isInteger(num2) && num1 > 0 && num2 > 0) {
  // Find the HCF or GCD
  let smallerNumber = Math.min(num1, num2);
  let hcf = 1;

  for (let i = 1; i <= smallerNumber; i++) {
    if (num1 % i === 0 && num2 % i === 0) {
      hcf = i;
    }
  }

  console.log(`The HCF (GCD) of ${num1} and ${num2} is: ${hcf}`);
} else {
  console.log("Please enter valid positive integers.");
}
```

24. Find LCM

This program prompts the user to enter two positive integers and calculates their Least Common Multiple (LCM).

```
// Prompt user for two positive integers
let num1 = parseInt(prompt("Enter the first positive integer:"));
let num2 = parseInt(prompt("Enter the second positive integer:"));

// Check if inputs are valid positive integers
if (!isNaN(num1) && !isNaN(num2) && Number.isInteger(num1) && Number.isInteger(num2) && num1 > 0 &&
num2 > 0) {
  // Find the LCM
  let largerNumber = Math.max(num1, num2);
  let lcm = largerNumber;

  while (true) {
```

```
    if (lcm % num1 === 0 && lcm % num2 === 0) {
      console.log(`The LCM of ${num1} and ${num2} is: ${lcm}`);
      break;
    }
    lcm += largerNumber;
  }
} else {
  console.log("Please enter valid positive integers.");
}
```

25. Find the Factors of a Number

This program prompts the user to enter a positive integer and calculates and displays its factors.

```
// Prompt user for a positive integer
let number = parseInt(prompt("Enter a positive integer:"));

// Check if input is a valid positive integer
if (!isNaN(number) && Number.isInteger(number) && number > 0) {
  console.log(`Factors of ${number}:`);

  // Find and display the factors
  for (let i = 1; i <= number; i++) {
    if (number % i === 0) {
      console.log(i);
    }
  }
} else {
  console.log("Please enter a valid positive integer.");
}
```

26. Find Sum of Natural Numbers Using Recursion

This program calculates the sum of natural numbers up to a given positive integer using recursion. It prompts the user to enter a positive integer n and then calculates and displays the sum using a recursive function.

```
// Define a recursive function to calculate the sum of natural numbers
function sumOfNaturalNumbers(n) {
  if (n === 1) {
    return 1;
  } else {
    return n + sumOfNaturalNumbers(n - 1);
  }
}

// Prompt user for a positive integer
let number = parseInt(prompt("Enter a positive integer:"));

// Check if input is a valid positive integer
if (!isNaN(number) && Number.isInteger(number) && number > 0) {
  // Calculate and display the sum using recursion
  let sum = sumOfNaturalNumbers(number);
  console.log(`The sum of natural numbers up to ${number} is: ${sum}`);
} else {
  console.log("Please enter a valid positive integer.");
```

}

27. Guess a Random Number

This program generates a random number between a specified range and allows the user to guess the number. It provides feedback on whether the guessed number is too high, too low, or correct. The user continues guessing until they correctly identify the random number.

```
// Generate a random number between 1 and 100 (you can adjust the range)
const randomNumber = Math.floor(Math.random() * 100) + 1;

// Initialize variables
let userGuess;
let attempts = 0;

// Game loop
do {
  // Prompt user for a guess
  userGuess = parseInt(prompt("Guess the random number (between 1 and 100):"));

  // Check if the input is a valid number
  if (!isNaN(userGuess)) {
    attempts++;

    // Provide feedback to the user
    if (userGuess < randomNumber) {
      console.log("Too low! Try again.");
    } else if (userGuess > randomNumber) {
```

```
      console.log("Too high! Try again.");
    } else {
      console.log(`Congratulations! You guessed the correct number ${randomNumber} in ${attempts} attempts.`);
    }
  } else {
    console.log("Please enter a valid number.");
  }
} while (userGuess !== randomNumber);
```

28. Shuffle Deck of Cards

This program simulates shuffling a standard deck of playing cards. It generates a deck of cards, displays the initial order, shuffles the deck, and then displays the shuffled order.

```
// Function to create a standard deck of cards
function createDeck() {
  const suits = ["Hearts", "Diamonds", "Clubs", "Spades"];
  const ranks = ["2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"];

  let deck = [];

  for (let suit of suits) {
    for (let rank of ranks) {
```

```
      deck.push(`${rank} of ${suit}`);
    }
  }

  return deck;
}

// Function to shuffle the deck of cards
function shuffleDeck(deck) {
  for (let i = deck.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [deck[i], deck[j]] = [deck[j], deck[i]]; // Swap elements to shuffle
  }
}

// Create and display an initial deck of cards
const initialDeck = createDeck();
console.log("Initial Deck:");
console.log(initialDeck);

// Shuffle the deck and display the shuffled deck
const shuffledDeck = [...initialDeck]; // Create a copy to avoid modifying the original
shuffleDeck(shuffledDeck);
console.log("\nShuffled Deck:");
console.log(shuffledDeck);
```

29. Display Fibonacci Sequence Using Recursion

This program displays the Fibonacci sequence up to a specified number of terms using recursion. It prompts the user to enter the number of terms and then uses a recursive function to generate and display the Fibonacci sequence.

```
// Function to generate the Fibonacci sequence using recursion
function fibonacci(n) {
  if (n <= 1) {
    return n;
  } else {
    return fibonacci(n - 1) + fibonacci(n - 2);
  }
}

// Prompt user for the number of terms in the Fibonacci sequence
let numTerms = parseInt(prompt("Enter the number of terms in the Fibonacci sequence:"));

// Check if input is a valid non-negative integer
if (!isNaN(numTerms) && Number.isInteger(numTerms) && numTerms >= 0) {
  console.log(`Fibonacci sequence of ${numTerms} terms:`);

  // Display the Fibonacci sequence using recursion
  for (let i = 0; i < numTerms; i++) {
    console.log(fibonacci(i));
  }
```

```
} else {
   console.log("Please enter a valid non-negative integer for the number of terms.");
}
```

30. Find Factorial of Number Using Recursion

This program calculates the factorial of a given non-negative integer using recursion. It prompts the user to enter a non-negative integer and then uses a recursive function to compute and display its factorial.

```
// Function to calculate the factorial using recursion
function factorial(n) {
  if (n === 0 || n === 1) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}

// Prompt user for a non-negative integer
let number = parseInt(prompt("Enter a non-negative integer:"));

// Check if input is a valid non-negative integer
if (!isNaN(number) && Number.isInteger(number) && number >= 0) {
  // Calculate and display the factorial using recursion
  let result = factorial(number);
  console.log(`The factorial of ${number} is: ${result}`);
} else {
  console.log("Please enter a valid non-negative integer.");
}
```

31. Convert Decimal to Binary

This program converts a decimal number to its binary equivalent. It prompts the user to enter a decimal number and then calculates and displays its binary representation.

```
// Function to convert decimal to binary
function decimalToBinary(decimalNumber) {
  if (decimalNumber === 0) {
    return "0";
  }

  let binaryResult = "";
  while (decimalNumber > 0) {
    let remainder = decimalNumber % 2;
    binaryResult = remainder + binaryResult;
    decimalNumber = Math.floor(decimalNumber / 2);
  }
```

```
  return binaryResult;
}

// Prompt user for a decimal number
let decimalNumber = parseInt(prompt("Enter a decimal number:"));

// Check if input is a valid integer
if (!isNaN(decimalNumber) && Number.isInteger(decimalNumber) && decimalNumber >= 0) {
  // Convert and display the binary equivalent
  let binaryEquivalent = decimalToBinary(decimalNumber);
  console.log(`The binary equivalent of ${decimalNumber} is: ${binaryEquivalent}`);
} else {
  console.log("Please enter a valid non-negative integer.");
}
```

32. Find ASCII Value of Character

This program prompts the user to enter a character and then calculates and displays its ASCII value.

```
// Prompt user for a character
let character = prompt("Enter a character:");
```

```
// Check if input is a valid single character
if (character.length === 1) {
   // Calculate and display the ASCII value
   let asciiValue = character.charCodeAt(0);
   console.log(`The ASCII value of '${character}' is: ${asciiValue}`);
} else {
   console.log("Please enter a valid single character.");
}
```

33. Check Whether a String is Palindrome or Not

This program checks whether a given string is a palindrome or not. A palindrome is a string that reads the same forward as backward.

```
// Function to check if a string is a palindrome
function isPalindrome(str) {
  // Remove non-alphanumeric characters and convert to lowercase
  const cleanStr = str.replace(/[^a-zA-Z0-9]/g, '').toLowerCase();

  // Compare the original and reversed strings
  return cleanStr === cleanStr.split('').reverse().join('');
}

// Prompt user for a string
let inputString = prompt("Enter a string:");

// Check if input is a valid string
if (inputString.length > 0) {
  // Check and display if the string is a palindrome
  if (isPalindrome(inputString)) {
    console.log(`"${inputString}" is a palindrome.`);
  } else {
    console.log(`"${inputString}" is not a palindrome.`);
  }
} else {
```

```
    console.log("Please enter a valid string.");
}
```

34. Sort Words in Alphabetical Order

This program prompts the user to enter a sentence or a list of words and then sorts and displays the words in alphabetical order.

```
// Prompt user for a sentence or a list of words
let inputString = prompt("Enter a sentence or a list of words:");

// Check if input is a valid string
if (inputString.length > 0) {
  // Split the input into an array of words
  let wordsArray = inputString.split(/\s+/);

  // Sort the array of words in alphabetical order
  let sortedWords = wordsArray.sort();

  // Display the sorted words
  console.log("Sorted Words:");
  console.log(sortedWords.join(', '));
} else {
  console.log("Please enter a valid string.");
}
```

35. Replace Characters of a String

This program prompts the user to enter a string, a target character, and a replacement character. It then replaces all occurrences of the target character with the replacement character and displays the modified string.

```
// Prompt user for a string
let inputString = prompt("Enter a string:");

// Check if input is a valid string
if (inputString.length > 0) {
  // Prompt user for a target character and a replacement character
  let targetChar = prompt("Enter the target character:");
  let replacementChar = prompt("Enter the replacement character:");

  // Check if targetChar is a single character and input is not empty
  if (targetChar.length === 1) {
    // Replace all occurrences of targetChar with replacementChar
    let modifiedString = inputString.split(targetChar).join(replacementChar);

    // Display the modified string
    console.log(`Modified String: ${modifiedString}`);
  } else {
    console.log("Please enter a valid target character (single character).");
  }
} else {
```

```
    console.log("Please enter a valid string.");
}
```

36. Reverse a String

This program prompts the user to enter a string and then reverses and displays the characters of the string.

```
// Prompt user for a string
let inputString = prompt("Enter a string:");

// Check if input is a valid string
if (inputString.length > 0) {
  // Reverse the string
  let reversedString = inputString.split('').reverse().join('');

  // Display the reversed string
  console.log(`Reversed String: ${reversedString}`);
} else {
  console.log("Please enter a valid string.");
}
```

37. Check the Number of Occurrences of a Character in the String

This program checks the number of occurrences of a character in a given string:

```
// Prompt user for a string and a character
let inputString = prompt("Enter a string:");
let targetChar = prompt("Enter the character to count:");

// Check if input is valid
if (inputString.length > 0 && targetChar.length === 1) {
  // Count occurrences of the target character
  let count = 0;
  for (let i = 0; i < inputString.length; i++) {
    if (inputString[i] === targetChar) {
      count++;
    }
  }

  // Display the result
  console.log(`Number of occurrences of '${targetChar}' in '${inputString}': ${count}`);
} else {
  console.log("Please enter a valid string and a single character.");
}
```

38. Convert the First Letter of a String into UpperCase

This program converts the first letter of a string to uppercase:

```
// Prompt user for a string
let inputString = prompt("Enter a string:");

// Check if input is a valid string
if (inputString.length > 0) {
  // Convert the first letter to uppercase
  let resultString = inputString.charAt(0).toUpperCase() + inputString.slice(1);

  // Display the result
  console.log(`Original String: ${inputString}`);
  console.log(`String with First Letter Uppercase: ${resultString}`);
} else {
  console.log("Please enter a valid string.");
}
```

39. Count the Number of Vowels in a String

This program counts the number of vowels in a given string:

```
// Prompt user for a string
let inputString = prompt("Enter a string:");

// Check if input is a valid string
if (inputString.length > 0) {
  // Function to count vowels in a string
  function countVowels(str) {
    const vowels = "aeiouAEIOU";
    let vowelCount = 0;

    for (let char of str) {
      if (vowels.includes(char)) {
        vowelCount++;
      }
    }

    return vowelCount;
}

// Call the function and display the result
let numberOfVowels = countVowels(inputString);
console.log(`Number of vowels in '${inputString}': ${numberOfVowels}`);
```

```
} else {
   console.log("Please enter a valid string.");
}
```

40. Check Whether a String Starts and Ends With Certain Characters

This program checks whether a string starts and ends with certain characters using the startsWith and endsWith methods in JavaScript:

```
// Prompt user for a string
let inputString = prompt("Enter a string:");

// Check if input is a valid string
if (inputString.length > 0) {
  // Prompt user for start and end characters
  let startChar = prompt("Enter the starting characters:");
  let endChar = prompt("Enter the ending characters:");

  // Check if the string starts and ends with the specified characters
  if (inputString.startsWith(startChar) && inputString.endsWith(endChar)) {
    console.log(`The string '${inputString}' starts with '${startChar}' and ends with '${endChar}'.`);
  } else {
    console.log(`The string '${inputString}' does not start with '${startChar}' or does not end with '${endChar}'.`);
  }
} else {
  console.log("Please enter a valid string.");
}
```

41. Replace All Occurrences of a String

Replace all occurrences of a substring in a string in JavaScript, you can use the `replace()` method along with a regular expression with the global (`g`) flag.

```
// Example string
let originalString = "Hello world, world!";

// String to replace
let searchString = "world";

// Replacement string
let replacementString = "universe";

// Replace all occurrences using replace() with a global regex
let modifiedString = originalString.replace(new RegExp(searchString, 'g'), replacementString);

// Display the result
console.log(`Original String: ${originalString}`);
console.log(`Modified String: ${modifiedString}`);
```

42. Create Multiline Strings

This program creates multiline strings using template literals (introduced in ECMAScript 6).

```
// Multiline string using template literals
let multilineString = `
  This is a multiline string.
  It spans multiple lines.
  You can include line breaks and indentation easily.
`;

console.log(multilineString);
```

43. Format Numbers as Currency Strings

This program format numbers as currency strings:

```
// Example number
let amount = 1234567.89;

// Format as currency string
let formattedAmount = amount.toLocaleString('en-US', { style: 'currency', currency: 'USD' });

// Display the formatted currency string
console.log(`Formatted Amount: ${formattedAmount}`);
```

44. Generate Random String

This program generates a random string in JavaScript, you can create a function that generates a string with random characters.

```
function generateRandomString(length) {
  const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
  let randomString = '';

  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * characters.length);
    randomString += characters.charAt(randomIndex);
  }

  return randomString;
}

// Generate a random string of length 8
let randomString = generateRandomString(8);

// Display the random string
console.log(`Random String: ${randomString}`);
```

45. Check if a String Starts with Another String

This program checks if a string starts with another string using the **startsWith** method.

```
// Example strings
let mainString = "Hello, World!";
let searchString = "Hello";

// Check if mainString starts with searchString
let startsWith = mainString.startsWith(searchString);

// Display the result
console.log(`Does the string start with '${searchString}'? ${startsWith}`);
```

46. Trim a String

This program trims leading and trailing whitespaces from a string using the **trim()** method.

```
// Example string with leading and trailing whitespaces
let stringWithSpaces = "   Hello, World!   ";

// Trim the string
let trimmedString = stringWithSpaces.trim();

// Display the result
console.log(`Original String: '${stringWithSpaces}'`);
console.log(`Trimmed String: '${trimmedString}'`);
```

47. Check Whether a String Contains a Substring

This program checks whether a string contains a substring in JavaScript using the **includes()** method or the **indexOf()** method.

```
// Example string
let mainString = "Hello, World!";
let substringToCheck = "World";

// Check if mainString contains substringToCheck
let containsSubstring = mainString.includes(substringToCheck);

// Display the result
console.log(`Does the string contain '${substringToCheck}'? ${containsSubstring}`);
```

48. Compare Two Strings

This program compare two strings in JavaScript, you can use the standard comparison operators (`==` , `!==` , `<` , `>` , `<=` , `>=`).

```
// Example strings
let string1 = "Hello";
let string2 = "hello";

// Case-sensitive comparison
let caseSensitiveComparison = string1 === string2;

// Case-insensitive comparison
let caseInsensitiveComparison = string1.toLowerCase() === string2.toLowerCase();

// Display the results
console.log(`Case-sensitive comparison: ${caseSensitiveComparison}`);
console.log(`Case-insensitive comparison: ${caseInsensitiveComparison}`);
```

49. Encode a String to Base64

This program encode a string to Base64 in JavaScript, you can use the **btoa()** function.

```
let originalString = "Hello, 你好 !";

// Encode the string to Base64 (for non-ASCII characters)
let base64EncodedString = btoa(encodeURIComponent(originalString));

// Display the result
console.log(`Original String: ${originalString}`);
console.log(`Base64 Encoded String: ${base64EncodedString}`);
```

50. Replace all Instances of a Character in a String

This program replace all instances of a character in a string in JavaScript, you can use the **replace()** method with a regular expression.

```
// Example string
let originalString = "Hello, World!";

// Character to replace
let charToReplace = "l";

// Replacement character
let replacementChar = "x";

// Replace all instances of charToReplace with replacementChar
let modifiedString = originalString.replace(new RegExp(charToReplace, 'g'), replacementChar);

// Display the result
console.log(`Original String: ${originalString}`);
console.log(`Modified String: ${modifiedString}`);
```

51. Replace All Line Breaks with

This program replace all line breaks (newline characters) in a string with a specific string or character, you can use the `replace()` method with a regular expression.

```
// Example string with line breaks
let stringWithLineBreaks = "Hello,\nWorld!\nThis is a new line.";

// Replacement string or character
let replacementString = "-";

// Replace all line breaks with the replacementString
let stringWithoutLineBreaks = stringWithLineBreaks.replace(/\n/g, replacementString);

// Display the result
console.log("Original String:");
console.log(stringWithLineBreaks);
console.log("\nString without Line Breaks:");
console.log(stringWithoutLineBreaks);
```

52. Check Leap Year

This program checks if a year is a leap year with a simple function. A leap year is a year that is evenly divisible by 4, except for years that are divisible by 100. However, years divisible by 400 are leap years.

```
function isLeapYear(year) {
  if ((year % 4 === 0 && year % 100 !== 0) || (year % 400 === 0)) {
    return true;
  } else {
    return false;
  }
}

// Example usage
let yearToCheck = 2024;
if (isLeapYear(yearToCheck)) {
  console.log(`${yearToCheck} is a leap year.`);
} else {
  console.log(`${yearToCheck} is not a leap year.`);
}
```

53. Format the Date

Formatting a date in JavaScript typically involves using the **`toLocaleDateString()`** method or constructing a formatted string manually. The **`toLocaleDateString()`** method provides a way to format a date based on the user's locale, including options for formatting the date, such as specifying the date style and time zone.

```
// Example date
let currentDate = new Date();

// Format the date using toLocaleDateString()
let formattedDate = currentDate.toLocaleDateString('en-US', {
  year: 'numeric',
  month: 'long',
  day: 'numeric',
  weekday: 'long',
  timeZone: 'UTC'
});

// Display the result
console.log(`Formatted Date: ${formattedDate}`);
```

54. Display Current Date

Display the current date in JavaScript, you can use the `Date` object to get the current date and time and then format it as needed.

```
// Get the current date
let currentDate = new Date();

// Format the current date as a string
let formattedDate = currentDate.toLocaleDateString('en-US');

// Display the result
console.log(`Current Date: ${formattedDate}`);
```

55. Compare The Value of Two Dates

Compare the values of two dates using standard comparison operators (`<` , `<=` , `>` , `>=`).

```
// Example dates
let date1 = new Date('2022-01-01');
let date2 = new Date('2023-01-01');

// Compare dates
if (date1 < date2) {
  console.log(`${date1} is earlier than ${date2}`);
} else if (date1 > date2) {
  console.log(`${date1} is later than ${date2}`);
} else {
  console.log(`${date1} is equal to ${date2}`);
}
```

56. Create Countdown Timer

Create a countdown timer in JavaScript, you can use the **setInterval()** function to update the timer at regular intervals and calculate the remaining time.

```
// Set the target date for the countdown
const targetDate = new Date('2023-01-01 00:00:00').getTime();

// Update the countdown every second
const countdownInterval = setInterval(updateCountdown, 1000);

function updateCountdown() {
  // Get the current date and time
  const currentDate = new Date().getTime();

  // Calculate the remaining time
  const timeDifference = targetDate - currentDate;

  // Calculate days, hours, minutes, and seconds
  const days = Math.floor(timeDifference / (1000 * 60 * 60 * 24));
  const hours = Math.floor((timeDifference % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60));
  const minutes = Math.floor((timeDifference % (1000 * 60 * 60)) / (1000 * 60));
  const seconds = Math.floor((timeDifference % (1000 * 60)) / 1000);

  // Display the countdown
  document.getElementById('countdown').innerHTML = `${days}d ${hours}h ${minutes}m ${seconds}s`;
```

```
    // Check if the countdown has reached zero
    if (timeDifference <= 0) {
        clearInterval(countdownInterval);
        document.getElementById('countdown').innerHTML = 'Countdown expired!';
    }
}
```

57. Remove Specific Item From an Array

Remove a specific item from an array in JavaScript, you can use the **filter()** method or the **splice()** method:

```
// Example array
let originalArray = [1, 2, 3, 4, 5];
let itemToRemove = 3;

// Find the index of the item to remove
let indexToRemove = originalArray.indexOf(itemToRemove);

// Remove the specific item using splice
if (indexToRemove !== -1) {
  originalArray.splice(indexToRemove, 1);
}

// Display the result
console.log("Original Array:", originalArray);
```

58. Check if An Array Contains a Specified Value

Check if an array contains a specified value in JavaScript, you can use the **includes()** method or the **indexOf()** method.

```
// Example array
let myArray = [1, 2, 3, 4, 5];
let valueToCheck = 3;

// Check if the array includes the specified value
let containsValue = myArray.includes(valueToCheck);

// Display the result
console.log(`Does the array include ${valueToCheck}? ${containsValue}`);
```

59. Insert Item in an Array

Insert an item into an array in JavaScript, you can use the `push()` method to add the item at the end of the array, or the `splice()` method to insert the item at a specific index.

```
// Example array
let myArray = [1, 2, 3, 4, 5];
let itemToInsert = 6;

// Insert the item at the end of the array using push
myArray.push(itemToInsert);

// Display the result
console.log("Array after inserting:", myArray);
```

60. Get Random Item From an Array

To get a random item from an array in JavaScript, you can use the **`Math.random()`** function to generate a random index within the array length.

```
// Example array
let myArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// Get a random index
let randomIndex = Math.floor(Math.random() * myArray.length);

// Get the random item from the array
let randomItem = myArray[randomIndex];

// Display the result
console.log("Random Item:", randomItem);
```

61. Perform Intersection Between Two Arrays

To find the intersection (common elements) between two arrays in JavaScript, you can use various methods.

```
// Example arrays
let array1 = [1, 2, 3, 4, 5];
let array2 = [3, 4, 5, 6, 7];

// Find the intersection using filter
let intersection = array1.filter(value => array2.includes(value));

// Display the result
console.log("Intersection:", intersection);
```

62. Split Array into Smaller Chunks

To split an array into smaller chunks in JavaScript, you can use a loop and the `slice()` method.

```
function chunkArray(array, chunkSize) {
  let result = [];
  for (let i = 0; i < array.length; i += chunkSize) {
    result.push(array.slice(i, i + chunkSize));
  }
  return result;
}

// Example array
let myArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// Split the array into chunks of size 3
let chunks = chunkArray(myArray, 3);

// Display the result
console.log("Original Array:", myArray);
console.log("Chunks:", chunks);
```

63. Get File Extension

To get the file extension from a string representing a file name in JavaScript, you can use various methods. One common approach is to use the **split()** method to split the file name based on the dot (**.**) and then extract the last part of the resulting array.

```
function getFileExtension(fileName) {
  // Split the file name based on the dot
  let parts = fileName.split('.');

  // Get the last part of the array (the file extension)
  let extension = parts[parts.length - 1];

  return extension;
}

// Example usage
let fileName = "example.txt";
let fileExtension = getFileExtension(fileName);

console.log(`File Extension: ${fileExtension}`);
```

64. Check If a Variable Is undefined or null

To check if a variable is `undefined` or `null` in JavaScript, you can use a simple conditional statement.

```
// Example variable
let myVariable = null;

// Check if the variable is undefined or null
if (myVariable === undefined || myVariable === null) {
  console.log("The variable is either undefined or null.");
} else {
  console.log("The variable is defined and not null.");
}
```

65. Generate a Random Number Between Two Numbers

To generate a random number between two specific numbers in JavaScript, you can use the **`Math.random()`** function along with some arithmetic.

```
function getRandomNumber(min, max) {
  // Generate a random decimal between 0 (inclusive) and 1 (exclusive)
  const randomDecimal = Math.random();

  // Scale the random decimal to the desired range
  const randomNumber = Math.floor(randomDecimal * (max - min + 1)) + min;

  return randomNumber;
}

// Example usage: Generate a random number between 1 and 100
let randomNum = getRandomNumber(1, 100);
console.log("Random Number:", randomNum);
```

66. Get The Current URL

To get the current URL in a web browser using JavaScript, you can use the **window.location** object.

```
// Get the domain
const domain = window.location.hostname;

// Get the path
const path = window.location.pathname;

// Get the query parameters
const queryParams = window.location.search;

// Display the results
console.log("Domain:", domain);
console.log("Path:", path);
console.log("Query Parameters:", queryParams);
```

67. Validate An Email Address

To validate an email address in JavaScript, you can use a regular expression.

```
function validateEmail(email) {
  // Regular expression for a basic email validation
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

  // Test the email against the regular expression
  return emailRegex.test(email);
}

// Example usage:
const emailToValidate = "example@email.com";
if (validateEmail(emailToValidate)) {
  console.log("Email is valid");
} else {
  console.log("Email is not valid");
}
```

68. Check If a Variable is of Function Type

To check if a variable is of the function type in JavaScript, you can use the `typeof` operator.

```
// Example variable
let myFunction = function() {
  console.log("Hello, world!");
};

// Check if the variable is a function
if (typeof myFunction === 'function') {
  console.log("The variable is of function type");
} else {
  console.log("The variable is not of function type");
}
```

69. Work With Constants

You can work with constants using the `const` keyword to declare variables whose values should not be reassigned.

```
// Define a constant
const PI = 3.14159;

// Attempt to reassign a constant (will result in an error)
// PI = 3.14; // Uncommenting this line will result in an error

// Display the value of the constant
console.log("The value of PI is:", PI);
```

70. Pass Parameter to a setTimeout() Function

To pass parameters to a function inside **setTimeout()**, you can use an anonymous function or an arrow function.

```
function myFunction(parameter) {
  console.log("Parameter received:", parameter);
}

// Using an anonymous function to pass parameters to setTimeout
let myParameter = "Hello, world!";
setTimeout(function() {
  myFunction(myParameter);
}, 1000);
```

71. Generate a Range of Numbers and Characters

To generate a range of numbers or characters in JavaScript, you can use a loop or other methods depending on your specific requirements.

```
function generateNumberRange(start, end) {
  let result = [];
  for (let i = start; i <= end; i++) {
    result.push(i);
  }
  return result;
}

// Example usage: Generate numbers from 1 to 5
let numberRange = generateNumberRange(1, 5);
console.log("Number Range:", numberRange);
```

72. Perform Function Overloading

JavaScript does not support traditional function overloading like some other programming languages do, where you can define multiple functions with the same name but different parameter types or counts. However, you can achieve similar behavior in JavaScript by checking the number or types of arguments within a function.

```
function exampleFunction() {
  if (arguments.length === 0) {
    // No arguments provided
    console.log("No arguments");
  } else if (arguments.length === 1 && typeof arguments[0] === "number") {
    // One argument of type number provided
    console.log("One number argument:", arguments[0]);
  } else if (arguments.length === 2 && typeof arguments[0] === "string" && typeof arguments[1] === "number") {
    // Two arguments: a string followed by a number
    console.log("String and number arguments:", arguments[0], arguments[1]);
  } else {
    // Default case
    console.log("Invalid arguments");
  }
}

// Example usage
```

```
exampleFunction();
exampleFunction(42);
exampleFunction("Hello", 7);
exampleFunction(true, "world"); // Invalid arguments
```

73. Implement a Stack

You can implement a basic stack data structure in JavaScript using an array.

```
class Stack {
  constructor() {
    this.items = [];
  }

  // Push an element onto the stack
  push(element) {
    this.items.push(element);
  }

// Pop the top element from the stack
pop() {
  if (this.isEmpty()) {
    return "Underflow";
  }
  return this.items.pop();
}

// Peek at the top element without removing it
peek() {
  return this.items[this.items.length - 1];
```

```
}

// Check if the stack is empty
isEmpty() {
  return this.items.length === 0;
}

// Get the size of the stack
size() {
  return this.items.length;
}

// Print the stack elements
  print() {
    console.log(this.items);
  }
}

// Example usage:
const stack = new Stack();

stack.push(1);
stack.push(2);
stack.push(3);

console.log("Stack elements:");
stack.print(); // Outputs: [1, 2, 3]
```

```
console.log("Top element:", stack.peek()); // Outputs: 3

console.log("Popped element:", stack.pop()); // Outputs: 3

console.log("Stack size:", stack.size()); // Outputs: 2

console.log("Is the stack empty?", stack.isEmpty()); // Outputs: false
```

74. Implement a Queue

You can implement a basic queue data structure in JavaScript using an array.

```
class Queue {
  constructor() {
    this.items = [];
  }

  // Enqueue an element at the end of the queue
```

```
enqueue(element) {
  this.items.push(element);
}

// Dequeue the element from the front of the queue
dequeue() {
  if (this.isEmpty()) {
    return "Underflow";
  }
  return this.items.shift();
}

// Peek at the front element without removing it
front() {
  if (this.isEmpty()) {
    return "Queue is empty";
  }
  return this.items[0];
}

// Check if the queue is empty
isEmpty() {
  return this.items.length === 0;
}

// Get the size of the queue
```

```
  size() {
    return this.items.length;
  }

  // Print the queue elements
  print() {
    console.log(this.items);
  }
}

// Example usage:
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(3);

console.log("Queue elements:");
queue.print(); // Outputs: [1, 2, 3]

console.log("Front element:", queue.front()); // Outputs: 1

console.log("Dequeued element:", queue.dequeue()); // Outputs: 1

console.log("Queue size:", queue.size()); // Outputs: 2

console.log("Is the queue empty?", queue.isEmpty()); // Outputs: false
```

75. Check if a Number is Float or Integer

You can check if a number is a float or an integer by examining its fractional part.

```
function checkNumberType(number) {
  // Check if the number has a fractional part
  if (Number.isInteger(number)) {
    console.log(number + " is an integer.");
  } else {
    console.log(number + " is a float.");
  }
}

// Example usage:
checkNumberType(5);      // Outputs: 5 is an integer.
checkNumberType(3.14);   // Outputs: 3.14 is a float.
checkNumberType(7.0);    // Outputs: 7 is an integer.
checkNumberType(-2.5);   // Outputs: -2.5 is a float.
checkNumberType(NaN);    // Outputs: NaN is not a valid number.
checkNumberType("abc");  // Outputs: abc is not a valid number.
```

76. Pass a Function as Parameter

You can pass a function as a parameter to another function. This allows you to create higher-order functions, which are functions that can accept other functions as arguments.

```
// Function that takes another function as a parameter
function operateOnNumbers(a, b, operation) {
  return operation(a, b);
}

// Example functions to be passed as parameters
function add(x, y) {
  return x + y;
}

function multiply(x, y) {
  return x * y;
}

// Example usage
let result1 = operateOnNumbers(3, 5, add);
console.log("Result of addition:", result1); // Outputs: 8

let result2 = operateOnNumbers(3, 5, multiply);
console.log("Result of multiplication:", result2); // Outputs: 15
```

77. Get the Dimensions of an Image

To get the dimensions (width and height) of an image in JavaScript, you can create an `Image` object, load the image, and then access the `width` and `height` properties.

```
function getImageDimensions(imageSrc) {
  // Create an Image object
  var img = new Image();

  // Set the source of the image
  img.src = imageSrc;

  // Wait for the image to load
  img.onload = function() {
    // Access the width and height properties
    var width = img.width;
    var height = img.height;

    // Display the dimensions
    console.log("Width:", width);
    console.log("Height:", height);
  };

  // Handle image loading error
  img.onerror = function() {
    console.error("Error loading the image.");
```

```
    };
}

// Example usage
var imageUrl = "path/to/your/image.jpg";
getImageDimensions(imageUrl);
```

78. Remove All Whitespaces From a Text

To remove all whitespaces from a text in JavaScript, you can use a regular expression and the **replace** method.

```
function removeWhitespaces(inputText) {
  // Use a regular expression to replace all whitespaces with an empty string
  return inputText.replace(/\s/g, "");
}

// Example usage
let textWithWhitespaces = "This is a text with spaces";
let textWithoutWhitespaces = removeWhitespaces(textWithWhitespaces);

console.log("Original Text:", textWithWhitespaces);
console.log("Text without Whitespaces:", textWithoutWhitespaces);
```

79. Write to Console

Writing to the console in JavaScript is typically done using the **`console.log()`** method.

```
// Write a message to the console
console.log("Hello, world!");

// You can also log variables or expressions
let number = 42;
console.log("The answer is:", number);

// Multiple values can be logged in a single statement
let firstName = "John";
let lastName = "Doe";
console.log("Full Name:", firstName, lastName);
```

80. Convert Date to Number

To convert a JavaScript `Date` object to a numeric value representing the number of milliseconds since the Unix Epoch (January 1, 1970), you can use the `getTime()` method.

```
// Create a Date object
const currentDate = new Date();

// Get the numeric representation of the date (milliseconds since the Unix Epoch)
const numericDate = currentDate.getTime();

console.log("Current Date:", currentDate);
console.log("Numeric Representation:", numericDate);
```

81. Find the Average of Two Numbers

To find the average of two numbers in JavaScript, you can add the two numbers together and then divide the sum by 2.

```
function findAverage(num1, num2) {
  // Calculate the sum of the two numbers
  let sum = num1 + num2;

  // Calculate the average by dividing the sum by 2
  let average = sum / 2;

  return average;
}

// Example usage:
let number1 = 10;
let number2 = 20;

let result = findAverage(number1, number2);

console.log(`The average of ${number1} and ${number2} is: ${result}`);
```

82. Calculate the Area of a Circle

This program calculate the area of a circle in JavaScript:

```
function calculateCircleArea(radius) {
  // Check if the radius is a valid number
  if (isNaN(radius) || radius <= 0) {
    return "Invalid radius. Please provide a positive number.";
  }

  // Calculate the area
  const area = Math.PI * Math.pow(radius, 2);

  return area;
}

// Example usage:
const radius = 5;
const area = calculateCircleArea(radius);

console.log(`The area of a circle with radius ${radius} is: ${area}`);
```

83. Random Color Generator

To generate a random color in JavaScript, you can create a function that generates random values for the red, green, and blue components of the color.

```
function generateRandomColor() {
  // Generate random values for red, green, and blue components
  const red = Math.floor(Math.random() * 256);
  const green = Math.floor(Math.random() * 256);
  const blue = Math.floor(Math.random() * 256);

  // Create the RGB color string
  const color = `rgb(${red}, ${green}, ${blue})`;

  return color;
}

// Example usage:
const randomColor = generateRandomColor();

console.log("Random Color:", randomColor);
```

84. Check if a String is Empty

To check if a string is empty in JavaScript, you can use the `length` property of the string. If the length is zero, it means the string is empty.

```
function isEmptyString(str) {
  return str.length === 0;
}

// Example usage:
const emptyString = "";
const nonEmptyString = "Hello, world!";

console.log("Is emptyString empty?", isEmptyString(emptyString)); // Outputs: true
console.log("Is nonEmptyString empty?", isEmptyString(nonEmptyString)); // Outputs: false
```

85. Capitalize the First Letter of a String

To capitalize the first letter of a string in JavaScript, you can use a combination of the **charAt()**, **toUpperCase()**, and **slice()** methods.

```
function capitalizeFirstLetter(str) {
  // Check if the string is not empty
  if (str.length === 0) {
    return "Empty string";
  }

  // Capitalize the first letter and concatenate the rest of the string
  return str.charAt(0).toUpperCase() + str.slice(1);
}

// Example usage:
const originalString = "hello, world!";
const capitalizedString = capitalizeFirstLetter(originalString);

console.log("Original String:", originalString);
console.log("Capitalized String:", capitalizedString);
```

86. Find the Maximum Element in an Array

To find the maximum element in an array in JavaScript, you can use the **`Math.max()`** function along with the spread operator (**`...`**) to pass the array elements as individual arguments.

```
function findMaxElement(arr) {
  // Check if the array is not empty
  if (arr.length === 0) {
    return "Empty array";
  }

  // Use Math.max() with the spread operator to find the maximum element
  const maxElement = Math.max(...arr);

  return maxElement;
}

// Example usage:
const numbers = [5, 2, 9, 1, 7];
const maxNumber = findMaxElement(numbers);

console.log("Array:", numbers);
console.log("Maximum Element:", maxNumber);
```

87. Reverse an Array

To reverse an array in JavaScript, you can use the **`reverse()`** method, which is available for arrays. This method reverses the elements of an array in place.

```
function reverseArray(arr) {
  // Use the reverse() method to reverse the array in place
  return arr.reverse();
}

// Example usage:
const originalArray = [1, 2, 3, 4, 5];
const reversedArray = reverseArray(originalArray);

console.log("Original Array:", originalArray);
console.log("Reversed Array:", reversedArray);
```

88. Calculate the Power of a Number

To calculate the power of a number in JavaScript, you can use the **`Math.pow()`** method or the exponentiation operator (`**`).

```
// Using Math.pow()
function calculatePowerWithMathPow(base, exponent) {
  return Math.pow(base, exponent);
}

// Using the exponentiation operator (**)
function calculatePowerWithExponentiationOperator(base, exponent) {
  return base ** exponent;
}

// Example usage:
const baseNumber = 2;
const exponentNumber = 3;

const resultWithMathPow = calculatePowerWithMathPow(baseNumber, exponentNumber);
const resultWithExponentiationOperator = calculatePowerWithExponentiationOperator(baseNumber, exponentNumber);

console.log(`${baseNumber} to the power of ${exponentNumber} using Math.pow(): ${resultWithMathPow}`);
console.log(`${baseNumber} to the power of ${exponentNumber} using the exponentiation operator (**): ${resultWithExponentiationOperator}`);
```

89. Find the Minimum Element in an Array

To find the minimum element in an array in JavaScript, you can use the `Math.min()` function along with the spread operator (`...`) to pass the array elements as individual arguments.

```
function findMinElement(arr) {
  // Check if the array is not empty
  if (arr.length === 0) {
    return "Empty array";
  }

  // Use Math.min() with the spread operator to find the minimum element
  const minElement = Math.min(...arr);

  return minElement;
}

// Example usage:
const numbers = [5, 2, 9, 1, 7];
const minNumber = findMinElement(numbers);

console.log("Array:", numbers);
console.log("Minimum Element:", minNumber);
```

90. Convert Minutes to Hours and Minutes

To convert a total number of minutes to hours and remaining minutes in JavaScript, you can use simple mathematical operations.

```
function convertMinutesToHoursAndMinutes(totalMinutes) {
  // Check if the input is a valid positive number
  if (isNaN(totalMinutes) || totalMinutes < 0) {
    return "Invalid input. Please provide a non-negative number of minutes.";
  }

  // Calculate hours and remaining minutes
  const hours = Math.floor(totalMinutes / 60);
  const minutes = totalMinutes % 60;

  // Construct the result string
  const result = `${hours} hours and ${minutes} minutes`;

  return result;
}

// Example usage:
const totalMinutes = 135;
const convertedTime = convertMinutesToHoursAndMinutes(totalMinutes);

console.log(`${totalMinutes} minutes is equivalent to: ${convertedTime}`);
```

91. Find the Sum of Digits in a Number

To find the sum of digits in a number using JavaScript, you can use a loop to iterate through each digit and add them together.

```
function sumOfDigits(number) {
  // Check if the input is a valid number
  if (isNaN(number) || !Number.isInteger(number) || number < 0) {
    return "Invalid input. Please provide a non-negative integer.";
  }

  // Convert the number to a string to iterate through its digits
  const digitsArray = String(number).split('').map(Number);

  // Calculate the sum of digits
  const sum = digitsArray.reduce((acc, digit) => acc + digit, 0);

  return sum;
}

// Example usage:
const inputNumber = 12345;
const result = sumOfDigits(inputNumber);

console.log(`The sum of digits in ${inputNumber} is: ${result}`);
```

92. Check if a String is a Palindromic Phrase

To check if a string is a palindromic phrase in JavaScript, you can create a function that removes non-alphanumeric characters and compares the string with its reversed version.

```
function isPalindromicPhrase(str) {
  // Check if the input is a valid string
  if (typeof str !== 'string') {
    return "Invalid input. Please provide a string.";
  }

  // Remove non-alphanumeric characters and convert to lowercase
  const cleanedStr = str.replace(/[^a-zA-Z0-9]/g, '').toLowerCase();

  // Compare the cleaned string with its reversed version
  return cleanedStr === cleanedStr.split('').reverse().join('');
}

// Example usage:
const phrase1 = "A man, a plan, a canal, Panama!";
const phrase2 = "Hello, world!";

console.log(`Is "${phrase1}" a palindromic phrase? ${isPalindromicPhrase(phrase1)}`);
console.log(`Is "${phrase2}" a palindromic phrase? ${isPalindromicPhrase(phrase2)}`);
```

93. Generate a Random Password

To generate a random password in JavaScript, you can create a function that combines random characters from various character sets.

```
function generateRandomPassword(length) {
  // Define character sets
  const uppercaseChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  const lowercaseChars = 'abcdefghijklmnopqrstuvwxyz';
  const numericChars = '0123456789';
  const specialChars = '!@#$%^&*()-_+=';

  // Combine character sets
  const allChars = uppercaseChars + lowercaseChars + numericChars + specialChars;

  // Check if the input length is a valid positive number
  if (!Number.isInteger(length) || length <= 0) {
    return "Invalid input. Please provide a positive integer for the password length.";
  }

  // Generate the random password
  let password = '';
  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * allChars.length);
    password += allChars.charAt(randomIndex);
```

```
  }

  return password;
}

// Example usage:
const passwordLength = 12;
const randomPassword = generateRandomPassword(passwordLength);

console.log(`Generated Password: ${randomPassword}`);
```

94. Calculate Simple Interest

Program To calculate simple interest in JavaScript:

```
function calculateSimpleInterest(principal, rate, time) {
  // Check if the inputs are valid positive numbers
  if (isNaN(principal) || isNaN(rate) || isNaN(time) || principal <= 0 || rate <= 0 || time <= 0) {
    return "Invalid inputs. Please provide valid positive numbers.";
  }

  // Calculate simple interest
  const simpleInterest = (principal * rate * time) / 100;

  return simpleInterest;
}

// Example usage:
const principalAmount = 1000;
const interestRate = 5; // 5%
const investmentTime = 2; // 2 years

const interestAmount = calculateSimpleInterest(principalAmount, interestRate, investmentTime);

console.log(`Principal Amount: $${principalAmount}`);
console.log(`Interest Rate: ${interestRate}%`);
console.log(`Investment Time: ${investmentTime} years`);
```

```
console.log(`Simple Interest: ${interestAmount}`);
```

95. Implement a Basic Stopwatch

To implement a basic stopwatch in JavaScript, you can use the `Date` object to measure the elapsed time.

```
let startTime;
let stopwatchInterval;

function startStopwatch() {
  startTime = new Date().getTime();

  stopwatchInterval = setInterval(updateDisplay, 1000);
}

function stopStopwatch() {
  clearInterval(stopwatchInterval);
}

function resetStopwatch() {
  stopStopwatch();
  updateDisplay(0);
}

function updateDisplay() {
  const currentTime = new Date().getTime();
  const elapsedTime = Math.floor((currentTime - startTime) / 1000);

  const minutes = Math.floor(elapsedTime / 60);
```

```
  const seconds = elapsedTime % 60;

  const formattedTime = `${minutes}:${seconds < 10 ? "0" : ""}${seconds}`;

  document.getElementById("display").textContent = formattedTime;
}
```

96. Check if a Number is a Perfect Number

A perfect number is a positive integer that is equal to the sum of its proper divisors (excluding itself).

```
function isPerfectNumber(number) {
  // Check if the input is a positive integer
  if (!Number.isInteger(number) || number <= 0) {
    return "Invalid input. Please provide a positive integer.";
  }

  // Find divisors and calculate sum
  let sum = 0;
  for (let i = 1; i <= Math.floor(number / 2); i++) {
    if (number % i === 0) {
      sum += i;
    }
  }

  // Check if the sum of divisors equals the original number
  return sum === number;
}

// Example usage:
const testNumber = 28;
const result = isPerfectNumber(testNumber);
```

```
console.log(`Is ${testNumber} a perfect number? ${result}`);
```

97. This program calculates the Volume of a Cylinder

This program calculates the Volume of a Cylinder:

```
function calculateCylinderVolume(radius, height) {
  // Check if the inputs are valid positive numbers
  if (isNaN(radius) || isNaN(height) || radius <= 0 || height <= 0) {
    return "Invalid inputs. Please provide valid positive numbers.";
  }

  // Calculate the volume of the cylinder
  const volume = Math.PI * Math.pow(radius, 2) * height;

  return volume;
}
```

```
// Example usage:
const cylinderRadius = 5;
const cylinderHeight = 10;

const cylinderVolume = calculateCylinderVolume(cylinderRadius, cylinderHeight);

console.log(`Cylinder Volume: ${cylinderVolume.toFixed(2)} cubic units`);
```

98. Generate a Random Quote

To generate a random quote in JavaScript, you can create an array of quotes and use a function to pick a random quote from that array.

```
function generateRandomQuote() {
  const quotes = [
    "The only way to do great work is to love what you do. - Steve Jobs",
    "In three words I can sum up everything I've learned about life: it goes on. - Robert Frost",
    "The greatest glory in living lies not in never falling, but in rising every time we fall. - Nelson Mandela",
    "Life is what happens when you're busy making other plans. - John Lennon",
    "Get busy living or get busy dying. - Stephen King"
    // Add more quotes as needed
  ];

  // Generate a random index to pick a quote from the array
  const randomIndex = Math.floor(Math.random() * quotes.length);

  // Return the randomly selected quote
  return quotes[randomIndex];
}

// Example usage:
const randomQuote = generateRandomQuote();
console.log("Random Quote:", randomQuote);
```

99. Find the Intersection of Two Arrays

To find the intersection of two arrays in JavaScript, you can create a function that iterates through both arrays and identifies the common elements.

```
function findIntersection(arr1, arr2) {
  // Check if the inputs are valid arrays
  if (!Array.isArray(arr1) || !Array.isArray(arr2)) {
    return "Invalid inputs. Please provide valid arrays.";
  }

  // Use a Set to store unique elements of the first array
  const set = new Set(arr1);

  // Filter the second array to include only elements present in the set
  const intersection = arr2.filter(element => set.has(element));

  return intersection;
}

// Example usage:
const array1 = [1, 2, 3, 4, 5];
const array2 = [3, 4, 5, 6, 7];

const result = findIntersection(array1, array2);
```

```
console.log("Intersection of Arrays:", result);
```

100. Convert Feet to Meters

To convert feet to meters in JavaScript, you can use the following conversion formula:

Meters=Feet×0.3048Meters=Feet×0.3048

Here's a simple function that performs the conversion:

```
function feetToMeters(feet) {
  // Check if the input is a valid number
  if (isNaN(feet)) {
    return "Invalid input. Please provide a valid number of feet.";
  }

  // Perform the conversion
  const meters = feet * 0.3048;

  return meters;
}

// Example usage:
const feetValue = 10;
const metersValue = feetToMeters(feetValue);

console.log(`${feetValue} feet is equal to ${metersValue.toFixed(2)} meters`);
```

101. Convert Days to Years, Months, and Days

To convert a given number of days into years, months, and remaining days in JavaScript, you can use the following function:

```
function convertDaysToYearsMonthsDays(days) {
  // Check if the input is a valid positive number
  if (isNaN(days) || days <= 0) {
    return "Invalid input. Please provide a valid positive number of days.";
  }

  // Calculate years
  const years = Math.floor(days / 365);

  // Calculate remaining days after removing years
  const remainingDaysAfterYears = days % 365;

  // Calculate months
  const months = Math.floor(remainingDaysAfterYears / 30);

  // Calculate remaining days after removing months
  const remainingDaysAfterMonths = remainingDaysAfterYears % 30;

  return {
    years,
    months,
```

```
    days: remainingDaysAfterMonths
  };
}

// Example usage:
const totalDays = 1000;
const result = convertDaysToYearsMonthsDays(totalDays);

console.log(`${totalDays} days is approximately ${result.years} years, ${result.months} months, and ${result.days} days.`);
```

102. Find the Median of an Array

To find the median of an array in JavaScript, you can create a function that sorts the array and then determines the median based on its length.

```
function findMedian(arr) {
  // Check if the input is a valid array
  if (!Array.isArray(arr) || arr.length === 0) {
    return "Invalid input. Please provide a non-empty array.";
  }

  // Sort the array
  const sortedArray = arr.slice().sort((a, b) => a - b);

  // Calculate the median
  const middleIndex = Math.floor(sortedArray.length / 2);

  if (sortedArray.length % 2 === 0) {
    // If the array has an even number of elements, return the average of the middle two
    const median = (sortedArray[middleIndex - 1] + sortedArray[middleIndex]) / 2;
    return median;
  } else {
    // If the array has an odd number of elements, return the middle element
    return sortedArray[middleIndex];
  }
```

```
}

// Example usage:
const numbers = [5, 2, 8, 1, 7, 3];
const result = findMedian(numbers);

console.log("Median:", result);
```

103. Calculate the Distance Between Two Points

To calculate the distance between two points (x1, y1) and (x2, y2) in a two-dimensional plane:

```
function calculateDistance(x1, y1, x2, y2) {
  // Check if the inputs are valid numbers
  if (isNaN(x1) || isNaN(y1) || isNaN(x2) || isNaN(y2)) {
    return "Invalid inputs. Please provide valid numerical coordinates.";
  }

  // Calculate the distance using the distance formula
  const distance = Math.sqrt(Math.pow((x2 - x1), 2) + Math.pow((y2 - y1), 2));

  return distance;
}
```

```
// Example usage:
const x1 = 1;
const y1 = 2;
const x2 = 4;
const y2 = 6;

const result = calculateDistance(x1, y1, x2, y2);

console.log(`The distance between (${x1}, ${y1}) and (${x2}, ${y2}) is ${result.toFixed(2)}`);
```

104. Check if a Number is a Perfect Square

To check if a number is a perfect square in JavaScript, you can calculate the square root of the number and then check if the square of the integer part of the square root is equal to the original number.

```
function isPerfectSquare(number) {
  // Check if the input is a valid positive number
  if (isNaN(number) || number < 0) {
    return "Invalid input. Please provide a valid non-negative number.";
  }

  // Calculate the square root
  const squareRoot = Math.sqrt(number);

  // Check if the square of the integer part of the square root is equal to the original number
  return squareRoot === Math.floor(squareRoot);
}

// Example usage:
const testNumber = 25;
const result = isPerfectSquare(testNumber);

console.log(`Is ${testNumber} a perfect square? ${result}`);
```

105. Find the Area of a Rectangle

To find the area of a rectangle in JavaScript, you can create a function that takes the length and width as parameters and applies the formula:

Area=Length×WidthArea=Length×Width

```
function calculateRectangleArea(length, width) {
  // Check if the inputs are valid positive numbers
  if (isNaN(length) || isNaN(width) || length <= 0 || width <= 0) {
    return "Invalid inputs. Please provide valid positive numbers for length and width.";
  }

  // Calculate the area of the rectangle
  const area = length * width;

  return area;
}

// Example usage:
const rectangleLength = 5;
const rectangleWidth = 8;

const result = calculateRectangleArea(rectangleLength, rectangleWidth);

console.log(`The area of the rectangle with length ${rectangleLength} and width ${rectangleWidth} is ${result}`);
```

106. Convert Binary to Decimal

To convert a binary number to a decimal number in JavaScript, you can create a function that takes a binary string as input and calculates the decimal equivalent.

```
function binaryToDecimal(binaryString) {
  // Check if the input is a valid binary string
  if (!/^[01]+$/.test(binaryString)) {
    return "Invalid input. Please provide a valid binary string.";
  }

  // Convert binary to decimal
  const decimalValue = parseInt(binaryString, 2);

  return decimalValue;
}

// Example usage:
const binaryNumber = "1101";
const decimalResult = binaryToDecimal(binaryNumber);

console.log(`The decimal equivalent of binary ${binaryNumber} is ${decimalResult}`);
```

107. Count the Number of Words in a Sentence

To count the number of words in a sentence in JavaScript, you can create a function that splits the sentence into words and then counts the number of resulting elements in the array.

```
function countWords(sentence) {
  // Check if the input is a valid string
  if (typeof sentence !== 'string') {
    return "Invalid input. Please provide a valid sentence.";
  }

  // Use a regular expression to split the sentence into words
  const words = sentence.split(/\s+/);

  // Filter out empty strings
  const nonEmptyWords = words.filter(word => word.length > 0);

  // Count the number of words
  const wordCount = nonEmptyWords.length;

  return wordCount;
}

// Example usage:
const sentence = "This is a sample sentence.";
const result = countWords(sentence);
```

```
console.log(`The sentence "${sentence}" has ${result} words.`);
```

108. Find the Union of Two Arrays

To find the union of two arrays in JavaScript, you can create a function that concatenates the arrays and removes duplicate elements.

```
function findUnion(arr1, arr2) {
  // Check if the inputs are valid arrays
  if (!Array.isArray(arr1) || !Array.isArray(arr2)) {
    return "Invalid inputs. Please provide valid arrays.";
  }

  // Concatenate the arrays and remove duplicates
  const unionArray = Array.from(new Set([...arr1, ...arr2]));

  return unionArray;
}

// Example usage:
const array1 = [1, 2, 3, 4, 5];
const array2 = [3, 4, 5, 6, 7];

const result = findUnion(array1, array2);

console.log("Union of Arrays:", result);
```

109. Calculate the Factorial of a Large Number

Calculating the factorial of a large number can quickly lead to very large results, and JavaScript has limitations in terms of precision. To handle large factorials, you can use libraries that support arbitrary-precision arithmetic. One such library is the "BigInt" type in JavaScript, which is available in modern environments.

```
function calculateFactorial(n) {
  // Check if the input is a valid non-negative integer
  if (!Number.isInteger(n) || n < 0) {
    return "Invalid input. Please provide a non-negative integer.";
  }

  // Use BigInt to handle large factorials
  let result = BigInt(1);
  for (let i = 2; i <= n; i++) {
    result *= BigInt(i);
  }

  return result.toString();
}

// Example usage:
const number = 50;
const result = calculateFactorial(number);
```

```
console.log(`The factorial of ${number} is:\n${result}`);
```

110. Check if a Number is a Strong Number

A strong number (or digital factorial) is a number such that the sum of its individual digits, each factorialized, is equal to the number itself. To check if a number is a strong number in JavaScript, you can create a function that calculates the factorial of each digit and checks if their sum equals the original number.

```
function calculateFactorial(number) {
  let result = 1;
  for (let i = 2; i <= number; i++) {
    result *= i;
  }
  return result;
}

function isStrongNumber(num) {
  // Check if the input is a non-negative integer
  if (!Number.isInteger(num) || num < 0) {
    return false;
  }

  // Convert the number to an array of digits
  const digits = Array.from(String(num), Number);

  // Calculate the factorial of each digit and sum them
```

```
    const digitFactorialSum = digits.reduce((sum, digit) => sum + calculateFactorial(digit), 0);

    // Check if the sum equals the original number
    return digitFactorialSum === num;
}

// Example usage:
const testNumber = 145;
const result = isStrongNumber(testNumber);

console.log(`${testNumber} is a strong number: ${result}`);
```

111. Check if a Number is a Narcissistic Number

A Narcissistic Number (or Armstrong Number) is a number that is the sum of its own digits each raised to the power of the number of digits. For example, 153 is a narcissistic number because 1^3 + 5^3 + 3^3 equals 153.

```
function isNarcissisticNumber(num) {
  // Check if the input is a non-negative integer
  if (!Number.isInteger(num) || num < 0) {
    return false;
  }

  const numStr = num.toString();
  const numDigits = numStr.length;
  let sum = 0;

  for (let i = 0; i < numDigits; i++) {
    const digit = parseInt(numStr[i], 10);
    sum += Math.pow(digit, numDigits);
  }

  return sum === num;
}

// Example usage:
const testNumber = 1634;
```

```
const result = isNarcissisticNumber(testNumber);

console.log(`${testNumber} is a Narcissistic Number: ${result}`);
```

112. Count the Number of Consonants in a String

You can create a JavaScript function to count the number of consonants in a given string.

```
function countConsonants(str) {
  // Check if the input is a valid string
  if (typeof str !== 'string') {
    return "Invalid input. Please provide a valid string.";
  }

  // Define a regular expression pattern for consonants
  const consonantPattern = /[bcdfghjklmnpqrstvwxyz]/gi;

  // Use match() to find all consonants in the string
  const consonants = str.match(consonantPattern);

  // Return the count of consonants
  return consonants ? consonants.length : 0;
}

// Example usage:
const testString = "Hello World";
const result = countConsonants(testString);

console.log(`The number of consonants in "${testString}" is: ${result}`);
```

113. Check if a Number is a Triangular Number

This program checks if a given positive integer is a triangular number. A triangular number is a number that can be represented in the form of a triangle with dots. Mathematically, a triangular number `T_n` is given by the formula: `T_n = n * (n + 1) / 2`.

```
function isTriangularNumber(num) {
  // Check if the input is a non-negative integer
  if (!Number.isInteger(num) || num < 0) {
    return false;
  }

  // Iterate through natural numbers until the sum exceeds or equals the input number
  let sum = 0;
  let n = 0;

  while (sum < num) {
    n++;
    sum += n;
  }

  // Check if the input number is equal to a triangular number
  return sum === num;
}

// Example usage:
```

```
const testNumber = 10;
const result = isTriangularNumber(testNumber);

console.log(`${testNumber} is a triangular number: ${result}`);
```

114. Find the Area of a Trapezoid

This program calculates the area of a trapezoid given the lengths of its bases (`a` and `b`) and its height (`h`). The formula for the area of a trapezoid is: `Area = (a + b) * h / 2` .

```
function trapezoidArea(base1, base2, height) {
  // Check if the inputs are valid numbers
  if (isNaN(base1) || isNaN(base2) || isNaN(height) || base1 <= 0 || base2 <= 0 || height <= 0) {
    return "Invalid input. Please provide valid numbers.";
  }

  // Calculate the area of the trapezoid
  const area = 0.5 * height * (base1 + base2);

  return area;
}

// Example usage:
const base1Length = 5;
const base2Length = 9;
const trapezoidHeight = 4;

const result = trapezoidArea(base1Length, base2Length, trapezoidHeight);
console.log(`The area of the trapezoid is: ${result}`);
```

115. Calculate the Greatest Common Divisor (GCD) of Two Numbers

The Greatest Common Divisor (GCD), also known as the Greatest Common Factor (GCF) or Highest Common Factor (HCF), is the largest positive integer that divides both numbers without leaving a remainder. There are different algorithms to calculate the GCD of two numbers, and one commonly used algorithm is the Euclidean Algorithm.

```
function calculateGCD(a, b) {
  // Check if the inputs are valid numbers
  if (!Number.isInteger(a) || !Number.isInteger(b) || a <= 0 || b <= 0) {
    return "Invalid input. Please provide valid positive integers.";
  }

  // Apply the Euclidean Algorithm to find the GCD
  while (b !== 0) {
    const temp = b;
    b = a % b;
    a = temp;
  }

  return a;
}
```

```
// Example usage:
const number1 = 48;
const number2 = 18;

const gcdResult = calculateGCD(number1, number2);
console.log(`The GCD of ${number1} and ${number2} is: ${gcdResult}`);
```

116. Check if a Number is a Fibonacci Number

A Fibonacci number is a number in the Fibonacci sequence, which is a series of numbers where each number is the sum of the two preceding ones. The sequence starts with 0 and 1.

```
function isFibonacciNumber(num) {
  // Check if the input is a non-negative integer
  if (!Number.isInteger(num) || num < 0) {
    return false;
  }

  // Check if the number is a perfect square
  const isPerfectSquare = (n) => {
    const sqrt = Math.sqrt(n);
    return sqrt === Math.floor(sqrt);
  };

  // A number is a Fibonacci number if and only if one of (5 * num^2 + 4) or (5 * num^2 - 4) is a perfect square
  return isPerfectSquare(5 * num * num + 4) || isPerfectSquare(5 * num * num - 4);
}

// Example usage:
const testNumber = 8;
const result = isFibonacciNumber(testNumber);

console.log(`${testNumber} is a Fibonacci number: ${result}`);
```

117. Find the Perimeter of a Rectangle

This program calculates the perimeter of a rectangle given the lengths of its sides (**length** and **width**). The formula for the perimeter of a rectangle is: **Perimeter = 2 * (length + width)** .

```
function rectanglePerimeter(length, width) {
  // Check if the inputs are valid numbers
  if (isNaN(length) || isNaN(width) || length <= 0 || width <= 0) {
    return "Invalid input. Please provide valid positive numbers.";
  }

  // Calculate the perimeter of the rectangle
  const perimeter = 2 * length + 2 * width;

  return perimeter;
}

// Example usage:
const rectangleLength = 5;
const rectangleWidth = 8;

const perimeterResult = rectanglePerimeter(rectangleLength, rectangleWidth);
console.log(`The perimeter of the rectangle is: ${perimeterResult}`);
```

118. Convert Roman Numerals to Numbers

Converting Roman numerals to numbers involves understanding the Roman numeral system and its rules.

```
function romanToNumber(roman) {
  // Define a mapping of Roman numeral symbols to their numeric values
  const romanNumerals = {
    'I': 1,
    'V': 5,
    'X': 10,
    'L': 50,
    'C': 100,
    'D': 500,
    'M': 1000,
};

// Check if the input is a valid Roman numeral string
if (typeof roman !== 'string') {
  return "Invalid input. Please provide a valid Roman numeral string.";
}

// Convert the Roman numeral string to a numeric value
let result = 0;

for (let i = 0; i < roman.length; i++) {
```

```
    const currentSymbol = roman[i];
    const currentValue = romanNumerals[currentSymbol];
    const nextSymbol = roman[i + 1];
    const nextValue = romanNumerals[nextSymbol];

    // If the current value is less than the next value, subtract it; otherwise, add it
    if (nextValue > currentValue) {
      result -= currentValue;
    } else {
      result += currentValue;
    }
  }

  return result;
}

// Example usage:
const romanNumeral = 'XIV';
const numericValue = romanToNumber(romanNumeral);

console.log(`The numeric value of ${romanNumeral} is: ${numericValue}`);
```

119. Check if a String is Anagram of Another String

An anagram is a word or phrase formed by rearranging the letters of another. To check if two strings are anagrams, you can compare whether the characters in both strings have the same frequency.

```
function areAnagrams(str1, str2) {
  // Check if the inputs are valid strings
  if (typeof str1 !== 'string' || typeof str2 !== 'string') {
    return "Invalid input. Please provide valid strings.";
  }

  // Remove non-alphabetic characters and convert to lowercase for case-insensitive comparison
  const cleanStr1 = str1.replace(/[^a-zA-Z]/g, '').toLowerCase();
  const cleanStr2 = str2.replace(/[^a-zA-Z]/g, '').toLowerCase();

  // Check if the lengths are equal after removing non-alphabetic characters
  if (cleanStr1.length !== cleanStr2.length) {
    return false;
  }

  // Create character frequency maps for both strings
  const charFrequencyMap1 = {};
  const charFrequencyMap2 = {};

  // Update frequency maps for str1
  for (const char of cleanStr1) {
```

```
    charFrequencyMap1[char] = (charFrequencyMap1[char] || 0) + 1;
  }

  // Update frequency maps for str2
  for (const char of cleanStr2) {
    charFrequencyMap2[char] = (charFrequencyMap2[char] || 0) + 1;
  }

  // Check if the frequency maps are equal
  for (const char in charFrequencyMap1) {
    if (charFrequencyMap1[char] !== charFrequencyMap2[char]) {
      return false;
    }
  }

  return true;
}

// Example usage:
const string1 = 'listen';
const string2 = 'silent';

const result = areAnagrams(string1, string2);
console.log(`${string1} and ${string2} are anagrams: ${result}`);
```

120. Generate Pascal's Triangle

Pascal's Triangle is a triangular array of binomial coefficients. Each number in the triangle is the sum of the two directly above it.

```
function generatePascalsTriangle(numRows) {
  // Check if the input is a positive integer
  if (!Number.isInteger(numRows) || numRows <= 0) {
    return "Invalid input. Please provide a valid positive integer.";
  }

  const pascalsTriangle = [];

  for (let i = 0; i < numRows; i++) {
    const row = [];

    for (let j = 0; j <= i; j++) {
      // Calculate the binomial coefficient using nCr formula
      const coefficient = factorial(i) / (factorial(j) * factorial(i - j));
      row.push(coefficient);
    }

    pascalsTriangle.push(row);
  }

  return pascalsTriangle;
```

```
}

// Helper function to calculate factorial
function factorial(n) {
  if (n === 0 || n === 1) {
    return 1;
  }

  return n * factorial(n - 1);
}

// Example usage:
const numberOfRows = 5;
const triangle = generatePascalsTriangle(numberOfRows);

console.log(`Pascal's Triangle with ${numberOfRows} rows:`);
console.log(triangle);
```

121. Convert Decimal to Roman Numerals

Converting a decimal number to Roman numerals involves finding the combination of Roman numeral symbols that represent the given decimal value.

```
function decimalToRoman(num) {
  // Check if the input is a positive integer within the valid Roman numeral range
  if (!Number.isInteger(num) || num <= 0 || num > 3999) {
    return "Invalid input. Please provide a valid positive integer within the range 1 to 3999.";
  }

  // Define the Roman numeral symbols and their corresponding values
  const romanSymbols = [
    { value: 1000, symbol: 'M' },
    { value: 900, symbol: 'CM' },
    { value: 500, symbol: 'D' },
    { value: 400, symbol: 'CD' },
    { value: 100, symbol: 'C' },
    { value: 90, symbol: 'XC' },
    { value: 50, symbol: 'L' },
    { value: 40, symbol: 'XL' },
    { value: 10, symbol: 'X' },
    { value: 9, symbol: 'IX' },
    { value: 5, symbol: 'V' },
    { value: 4, symbol: 'IV' },
```

```
  { value: 1, symbol: 'I' },
];

// Convert the decimal number to Roman numerals
let result = '';

for (const symbol of romanSymbols) {
  while (num >= symbol.value) {
    result += symbol.symbol;
    num -= symbol.value;
  }
}

  return result;
}

// Example usage:
const decimalNumber = 1984;
const romanNumeral = decimalToRoman(decimalNumber);

console.log(`The Roman numeral representation of ${decimalNumber} is: ${romanNumeral}`);
```

122. Find the Area of a Parallelogram

This program calculates the area of a parallelogram given the length of its base and the corresponding perpendicular height.

```
function parallelogramArea(base, height) {
  // Check if the inputs are valid numbers
  if (isNaN(base) || isNaN(height) || base <= 0 || height <= 0) {
    return "Invalid input. Please provide valid positive numbers.";
  }

  // Calculate the area of the parallelogram
  const area = base * height;

  return area;
}

// Example usage:
const parallelogramBase = 6;
const parallelogramHeight = 8;

const areaResult = parallelogramArea(parallelogramBase, parallelogramHeight);
console.log(`The area of the parallelogram is: ${areaResult}`);
```

123. Basic Rock, Paper, Scissors Game

Here's a basic implementation of a Rock, Paper, Scissors game using JavaScript.

```
function computerPlay() {
  const options = ['rock', 'paper', 'scissors'];
  const randomIndex = Math.floor(Math.random() * options.length);
  return options[randomIndex];
}

function playRound(playerSelection, computerSelection) {
  playerSelection = playerSelection.toLowerCase();
  if (!['rock', 'paper', 'scissors'].includes(playerSelection)) {
    return 'Invalid choice. Please choose rock, paper, or scissors.';
  }

  console.log(`Player chooses ${playerSelection}`);
  console.log(`Computer chooses ${computerSelection}`);

  if (playerSelection === computerSelection) {
    return 'It\'s a tie!';
  } else if (
    (playerSelection === 'rock' && computerSelection === 'scissors') ||
    (playerSelection === 'paper' && computerSelection === 'rock') ||
    (playerSelection === 'scissors' && computerSelection === 'paper')
```

```
  ) {
    return 'You win!';
  } else {
    return 'You lose!';
  }
}

function game() {
  let playerScore = 0;
  let computerScore = 0;

  for (let i = 0; i < 5; i++) {
    const playerSelection = prompt('Enter your choice (rock, paper, or scissors):');
    const computerSelection = computerPlay();
    const result = playRound(playerSelection, computerSelection);

    console.log(result);

    if (result.includes('win')) {
      playerScore++;
    } else if (result.includes('lose')) {
      computerScore++;
    }
  }

  console.log(`Final Score: Player ${playerScore} - ${computerScore} Computer`);
```

```
  if (playerScore > computerScore) {
    console.log('You win the game!');
  } else if (playerScore < computerScore) {
    console.log('You lose the game!');
  } else {
    console.log('The game is a tie!');
  }
}

// Uncomment the line below to start the game
// game();
```

124. Basic Slot Machine Game

This program simulates a basic slot machine game in which three symbols are randomly selected. If all three symbols are the same, the player wins; otherwise, they are encouraged to try again.

```
// Array of symbols for the slot machine
const symbols = ['  ','  ','  ','  ','  ','  '];

// Function to get a random symbol from the array
function getRandomSymbol() {
  const randomIndex = Math.floor(Math.random() * symbols.length);
  return symbols[randomIndex];
}

// Function to simulate a spin of the slot machine
function spin() {
  // Get three random symbols
  const result1 = getRandomSymbol();
  const result2 = getRandomSymbol();
  const result3 = getRandomSymbol();

  // Display the results
  console.log(`Result: ${result1} - ${result2} - ${result3}`);

  // Check for a win
  if (result1 === result2 && result2 === result3) {
```

```
    console.log(' ��  Congratulations! You won!  ��  ');
  } else {
    console.log(' ��  Try again. Better luck next time!  ��  ');
  }
}

// Example usage - simulate a spin
spin();
```

125. Check if a Number is a Smith Number

This program checks if a given positive integer is a Smith number. A Smith number is a composite number for which the sum of its digits is equal to the sum of the prime factors' digits. The sum of the prime factors' digits is computed recursively until prime factors are obtained.

```
function isPrime(num) {
 if (num < 2) return false;
 for (let i = 2; i <= Math.sqrt(num); i++) {
  if (num % i === 0) return false;
 }
 return true;
}

function sumOfDigits(number) {
 return number.toString().split('').reduce((sum, digit) => sum + parseInt(digit, 10), 0);
}

function primeFactors(number) {
 const factors = [];
 for (let i = 2; i <= number; i++) {
  while (isPrime(i) && number % i === 0) {
   factors.push(i);
   number /= i;
  }
```

```
  }
  return factors;
}

function isSmithNumber(number) {
  const originalSum = sumOfDigits(number);
  const primeFactorSum = primeFactors(number).reduce((sum, factor) => sum + sumOfDigits(factor), 0);

  return originalSum === primeFactorSum;
}

// Example: Check if 728 is a Smith number
console.log(isSmithNumber(728)); // Output: true
```

126. Basic Chessboard

This program generates a basic representation of a chessboard using characters. It creates an 8x8 grid with alternating black and white squares, providing a simple visual representation of a chessboard.

```
// Function to generate a basic chessboard
function generateChessboard() {
  const size = 8; // Size of the chessboard (8x8)
  const chessboard = [];

  for (let row = 0; row < size; row++) {
    const currentRow = [];
```

```
    for (let col = 0; col < size; col++) {
      // Use 'X' for black squares and ' ' for white squares
      const isBlack = (row + col) % 2 === 1;
      const square = isBlack ? 'X' : ' ';
      currentRow.push(square);
    }

    chessboard.push(currentRow.join(' '));
  }

  return chessboard.join('\n');
}

// Example usage
const chessboard = generateChessboard();
console.log(chessboard);
```

127. Generate a Random Password with Specific Criteria

This program generates a random password based on specified criteria, including the length and the types of characters (uppercase letters, lowercase letters, numbers, and special characters).

```
// Function to generate a random password
function generateRandomPassword(length, includeUppercase, includeLowercase, includeNumbers, includeSpecialChars) {
  const uppercaseChars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  const lowercaseChars = 'abcdefghijklmnopqrstuvwxyz';
  const numberChars = '0123456789';
  const specialChars = '!@#$%^&*()-=_+[]{}|;:,.<>?';

  let validChars = '';
  let password = '';

  if (includeUppercase) validChars += uppercaseChars;
  if (includeLowercase) validChars += lowercaseChars;
  if (includeNumbers) validChars += numberChars;
  if (includeSpecialChars) validChars += specialChars;

  if (validChars.length === 0) {
   console.error('Error: At least one character type must be selected.');
   return null;
  }
```

```
  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * validChars.length);
    password += validChars.charAt(randomIndex);
  }

  return password;
}

// Example usage
const passwordLength = 12;
const includeUppercase = true;
const includeLowercase = true;
const includeNumbers = true;
const includeSpecialChars = true;

    const randomPassword = generateRandomPassword(passwordLength, includeUppercase, includeLowercase, includeNumbers, includeSpecialChars);

if (randomPassword) {
  console.log(`Random Password: ${randomPassword}`);
}
```

128. Basic Maze Solver

This program simulates a basic maze solver that navigates through a maze represented by a 2D array. The solver uses a simple depth-first search algorithm to find a path from the starting point to the exit.

```
// Function to solve a maze using depth-first search
function solveMaze(maze) {
  const rows = maze.length;
  const cols = maze[0].length;
  const visited = Array.from({ length: rows }, () => Array(cols).fill(false));

  function isSafe(row, col) {
   return row >= 0 && row < rows && col >= 0 && col < cols && maze[row][col] === 1 && !visited[row][col];
  }

  function dfs(row, col) {
   if (row === rows - 1 && col === cols - 1) {
    // Reached the exit
    return true;
   }

   if (isSafe(row, col)) {
    visited[row][col] = true;

    // Move in all possible directions (up, down, left, right)
    const directions = [
```

```
  [-1, 0], // Up
  [1, 0],  // Down
  [0, -1], // Left
  [0, 1]   // Right
];

for (const [dr, dc] of directions) {
  const newRow = row + dr;
  const newCol = col + dc;

  if (dfs(newRow, newCol)) {
    return true;
  }

  visited[row][col] = false; // Backtrack
 }

 return false;
}

// Start the depth-first search from the top-left corner
if (dfs(0, 0)) {
 console.log('Path found! ');
} else {
 console.log('No path found. ');
```

```
  }
}

// Example usage
const maze = [
  [1, 0, 1, 1, 1],
  [1, 1, 1, 0, 1],
  [0, 0, 1, 1, 1],
  [0, 0, 1, 0, 1],
  [0, 0, 1, 1, 1]
];

solveMaze(maze);
```

129. Check if a String is Pangram or Not

This program checks whether a given string is a pangram or not. A pangram is a sentence that contains every letter of the alphabet at least once.

```
// Function to check if a string is a pangram
function isPangram(str) {
  const alphabet = 'abcdefghijklmnopqrstuvwxyz';
  const lowercasedStr = str.toLowerCase();

  for (const char of alphabet) {
   if (!lowercasedStr.includes(char)) {
    return false;
   }
  }

  return true;
}

// Example usage
const inputString = "The quick brown fox jumps over the lazy dog";
if (isPangram(inputString)) {
  console.log('The given string is a pangram! ');
} else {
  console.log('The given string is not a pangram. ');
```

```
}
```

130. Basic Sudoku Solver

This program solves a basic 9x9 Sudoku puzzle using a backtracking algorithm. The solver fills in the empty cells of the Sudoku grid by recursively trying numbers until a solution is found.

```
// Function to check if a number can be placed in a particular cell
function isSafe(board, row, col, num) {
  // Check if 'num' is not present in the current row and column
  for (let i = 0; i < 9; i++) {
   if (board[row][i] === num || board[i][col] === num) {
    return false;
   }
  }

  // Check if 'num' is not present in the 3x3 subgrid
  const startRow = Math.floor(row / 3) * 3;
  const startCol = Math.floor(col / 3) * 3;
  for (let i = 0; i < 3; i++) {
   for (let j = 0; j < 3; j++) {
    if (board[startRow + i][startCol + j] === num) {
     return false;
    }
   }
  }
}
```

```
  return true;
}

// Function to find an empty cell in the Sudoku grid
function findEmptyCell(board) {
  for (let row = 0; row < 9; row++) {
    for (let col = 0; col < 9; col++) {
      if (board[row][col] === 0) {
        return [row, col];
      }
    }
  }
  return null;
}

// Function to solve the Sudoku puzzle using backtracking
function solveSudoku(board) {
  const emptyCell = findEmptyCell(board);

  if (!emptyCell) {
    // No empty cell found, the puzzle is solved
    return true;
  }

  const [row, col] = emptyCell;
```

```
// Try filling the empty cell with numbers 1 to 9
for (let num = 1; num <= 9; num++) {
 if (isSafe(board, row, col, num)) {
  // If placing 'num' in the current cell is safe, try the next step
  board[row][col] = num;

  // Recursively attempt to solve the rest of the puzzle
  if (solveSudoku(board)) {
   return true;
  }

  // If placing 'num' in the current cell leads to an invalid solution, backtrack
   board[row][col] = 0;
  }
 }

 // No number from 1 to 9 can be placed in the current cell, backtrack
 return false;
}

// Example usage
const sudokuBoard = [
 [5, 3, 0, 0, 7, 0, 0, 0, 0],
 [6, 0, 0, 1, 9, 5, 0, 0, 0],
 [0, 9, 8, 0, 0, 0, 0, 6, 0],
```

```
  [8, 0, 0, 0, 6, 0, 0, 0, 3],
  [4, 0, 0, 8, 0, 3, 0, 0, 1],
  [7, 0, 0, 0, 2, 0, 0, 0, 6],
  [0, 6, 0, 0, 0, 0, 2, 8, 0],
  [0, 0, 0, 4, 1, 9, 0, 0, 5],
  [0, 0, 0, 0, 8, 0, 0, 7, 9]
];

if (solveSudoku(sudokuBoard)) {
  console.log('Sudoku solved successfully:');
  console.log(sudokuBoard.map(row => row.join(' ')).join('\n'));
} else {
  console.log('No solution exists for the given Sudoku puzzle.');
}
```

131. Convert Objects to Strings

This program convert JavaScript objects to strings, you can use the **JSON.stringify()** method.

```
// Example object
let person = {
  firstName: "John",
  lastName: "Doe",
  age: 30
};

// Convert object to a JSON-formatted string
let jsonString = JSON.stringify(person);

// Display the result
console.log("Original Object:");
console.log(person);

console.log("\nJSON-formatted String:");
console.log(jsonString);
```

132. Find the Area of an Ellipse

This program calculates the area of an ellipse given its semi-major axis (a) and semi-minor axis (b). The formula for the area of an ellipse is $\pi \times a \times b$.

```
// Function to calculate the area of an ellipse
function calculateEllipseArea(semiMajorAxis, semiMinorAxis) {
  const pi = Math.PI;
  const area = pi * semiMajorAxis * semiMinorAxis;
  return area;
}

// Example usage
const semiMajorAxis = 5;
const semiMinorAxis = 3;

const ellipseArea = calculateEllipseArea(semiMajorAxis, semiMinorAxis);
console.log(`The area of the ellipse is: ${ellipseArea.toFixed(2)}`);
```

133. Check if a Number is a Palindrome in Binary

This program checks whether a given number is a palindrome in binary representation. A palindrome in binary reads the same backward as forward.

```
// Function to check if a number is a palindrome in binary
function isBinaryPalindrome(number) {
  const binaryRepresentation = number.toString(2);
  const reversedBinary = binaryRepresentation.split('').reverse().join('');

  return binaryRepresentation === reversedBinary;
}

// Example usage
const numberToCheck = 9; // Replace with the number you want to check
if (isBinaryPalindrome(numberToCheck)) {
  console.log(`${numberToCheck} is a binary palindrome! ✅`);
} else {
  console.log(`${numberToCheck} is not a binary palindrome. ❌`);
}
```

134. Find the Area of a Rhombus

This program calculates the area of a rhombus given the lengths of its diagonals (`d1` and `d2`). The formula for the area of a rhombus is `Area = (d1 * d2) / 2` , where `d1` and `d2` are the lengths of the diagonals.

```
// Function to calculate the area of a rhombus
function calculateRhombusArea(diagonal1, diagonal2) {
  const area = (diagonal1 * diagonal2) / 2;
  return area;
}

// Example usage
const diagonal1Length = 8;
const diagonal2Length = 6;

const rhombusArea = calculateRhombusArea(diagonal1Length, diagonal2Length);
console.log(`The area of the rhombus is: ${rhombusArea.toFixed(2)}`);
```

135. Check if a Number is a Catalan Number

This program checks if a given non-negative integer is a Catalan number. Catalan numbers are a sequence of natural numbers that occur in various counting problems, often involving recursive structures. The nth Catalan number is given by the formula: **Cn = (2n)! / ((n + 1)! * n!)** .

```
// Function to calculate binomial coefficient (n choose k)
function binomialCoefficient(n, k) {
 if (k > n - k) {
  k = n - k;
 }
 let result = 1;
 for (let i = 0; i < k; i++) {
  result *= (n - i);
  result /= (i + 1);
 }

 return result;
}

// Function to check if a number is a Catalan number
function isCatalanNumber(num) {
 for (let i = 0; i <= num; i++) {
  if (binomialCoefficient(2 * i, i) === num / (i + 1)) {
```

```
    return true;
   }
  }
  return false;
}

// Example usage
const numberToCheck = 42; // Replace with the number you want to check
if (isCatalanNumber(numberToCheck)) {
  console.log(`${numberToCheck} is a Catalan number! ✅`);
} else {
  console.log(`${numberToCheck} is not a Catalan number. ❌`);
}
```

136. Find the Luhn Algorithm Check Digit

The Luhn algorithm, also known as the modulus 10 or mod 10 algorithm, is a simple checksum formula used to validate various identification numbers, such as credit card numbers. The algorithm is used to detect accidental errors in these numbers.

```
function calculateLuhnCheckDigit(input) {
 // Convert the input string to an array of digits
 const digits = input.toString().split('').map(Number);

 // Double every second digit from the right
 for (let i = digits.length - 2; i >= 0; i -= 2) {
  let doubledDigit = digits[i] * 2;

  // If doubling results in a number greater than 9, subtract 9
  if (doubledDigit > 9) {
   doubledDigit -= 9;
  }

  digits[i] = doubledDigit;
 }

 // Calculate the sum of all digits
 const sum = digits.reduce((acc, digit) => acc + digit, 0);

 // Find the smallest number (x) such that (sum + x) is a multiple of 10
```

```
  const checkDigit = (10 - (sum % 10)) % 10;

  return checkDigit;
}

// Example usage
const partialNumber = 123456789; // Replace with the partial number for which you want to calculate the check digit
const checkDigit = calculateLuhnCheckDigit(partialNumber);
const fullNumber = `${partialNumber}${checkDigit}`;

console.log(`Partial Number: ${partialNumber}`);
console.log(`Check Digit: ${checkDigit}`);
console.log(`Full Number with Check Digit: ${fullNumber}`);
```

137. Basic Maze Generator

Creating a basic maze generator involves using an algorithm to generate a random maze layout. One common algorithm for maze generation is the recursive backtracking algorithm.

```
class MazeGenerator {
 constructor(rows, cols) {
  this.rows = rows;
  this.cols = cols;
  this.grid = Array.from({ length: rows }, () => Array(cols).fill(0));
 }

 generateMaze() {
  this.visitCell(0, 0);
}

visitCell(row, col) {
 this.grid[row][col] = 1; // Mark the current cell as visited

 // Define the possible directions (up, down, left, right)
 const directions = [
  [0, -2],
  [0, 2],
  [-2, 0],
  [2, 0]
```

```
    ];

    // Shuffle the directions randomly
    directions.sort(() => Math.random() - 0.5);

    for (const [dr, dc] of directions) {
      const newRow = row + dr;
      const newCol = col + dc;

      // Check if the new cell is within bounds and unvisited
      if (newRow >= 0 && newRow < this.rows && newCol >= 0 && newCol < this.cols && this.grid[newRow][newCol] ===
0) {
        // Mark the cell between the current and new cell as visited
        this.grid[row + dr / 2][col + dc / 2] = 1;

        // Recursively visit the new cell
        this.visitCell(newRow, newCol);
      }
    }
  }

  printMaze() {
    for (const row of this.grid) {
      console.log(row.map(cell => (cell === 1 ? ' ' : '#')).join(''));
    }
  }
```

```
}

// Example usage
const mazeRows = 11;
const mazeCols = 21;

const mazeGenerator = new MazeGenerator(mazeRows, mazeCols);
mazeGenerator.generateMaze();
mazeGenerator.printMaze();
```

138. Check if a Year is a Magic Year

A magic year is a year in which the product of the month and day (in two-digit form) equals the last two digits of the year. For example, in the year 1978, the product of the month (7) and the day (8) is 56, which matches the last two digits of the year.

```
function isMagicYear(year) {
  const yearStr = year.toString();
  const month = parseInt(yearStr.substring(0, 2), 10);
  const day = parseInt(yearStr.substring(2, 4), 10);

  return month * day === parseInt(yearStr.substring(4), 10);
}

// Example usage
const yearToCheck = 1978; // Replace with the year you want to check
if (isMagicYear(yearToCheck)) {
  console.log(`${yearToCheck} is a Magic Year! ��`);
} else {
  console.log(`${yearToCheck} is not a Magic Year. ��`);
}
```

139. Basic Crossword Puzzle

Creating a basic crossword puzzle involves defining a grid and placing words into it, ensuring that they intersect correctly. Below is a simple JavaScript program that generates a crossword puzzle with a few words. The generated puzzle is displayed in the console:

```
class CrosswordPuzzle {
 constructor(rows, cols) {
  this.rows = rows;
  this.cols = cols;
  this.grid = Array.from({ length: rows }, () => Array(cols).fill(' '));
  this.words = [];
 }

 addWord(word, row, col, direction) {
  if (direction === 'across') {
   for (let i = 0; i < word.length; i++) {
    this.grid[row][col + i] = word[i];
   }
  } else if (direction === 'down') {
   for (let i = 0; i < word.length; i++) {
    this.grid[row + i][col] = word[i];
   }
  }
```

```
    this.words.push({ word, row, col, direction });
  }

  printPuzzle() {
    for (const row of this.grid) {
      console.log(row.join(' '));
    }
  }
}

// Example usage
const crossword = new CrosswordPuzzle(5, 5);
crossword.addWord('apple', 1, 1, 'across');
crossword.addWord('banana', 3, 0, 'down');
crossword.addWord('grape', 0, 3, 'across');

crossword.printPuzzle();
```

140. Find the Area of a Regular Polygon

This program calculates the area of a regular polygon given the number of sides (`n`) and the side length (`s`). The formula for the area of a regular polygon is `Area = (n * s^2) / (4 * tan(π/n))` , where `n` is the number of sides, `s` is the side length, and `tan` is the tangent function.

```
function calculateRegularPolygonArea(n, s) {
  const numerator = 1 / 4 * n * Math.pow(s, 2);
  const denominator = Math.tan(Math.PI / n);
  const area = numerator / denominator;
  return area;
}

// Example usage
const numberOfSides = 6; // Replace with the number of sides of your polygon
const sideLength = 5;   // Replace with the length of each side of your polygon

const polygonArea = calculateRegularPolygonArea(numberOfSides, sideLength);
console.log(`The area of the regular polygon is: ${polygonArea.toFixed(2)}`);
```

141. Check if a Number is an Abundant Number

An abundant number (or excessive number) is a positive integer that is smaller than the sum of its proper divisors, excluding itself. The sum of proper divisors is greater than the number itself.

```
function getProperDivisorsSum(number) {
 let sum = 0;
 for (let i = 1; i <= Math.floor(number / 2); i++) {
  if (number % i === 0) {
   sum += i;
  }
 }
 return sum;
}

function isAbundantNumber(number) {
 const divisorsSum = getProperDivisorsSum(number);
 return divisorsSum > number;
}

// Example usage
const numberToCheck = 12; // Replace with the number you want to check
if (isAbundantNumber(numberToCheck)) {
 console.log(`${numberToCheck} is an Abundant Number! ✅`);
} else {
```

```
  console.log(`${numberToCheck} is not an Abundant Number. ⛔ `);
}
```

142. Basic Word Guessing Game

Creating a basic word guessing game in a console environment involves generating a random word, allowing the player to make guesses, and providing feedback on correct and incorrect guesses.

```
class WordGuessingGame {
 constructor(wordList) {
  this.wordList = wordList;
  this.selectedWord = this.getRandomWord();
  this.guesses = [];
  this.maxAttempts = 6;
 }

getRandomWord() {
 const randomIndex = Math.floor(Math.random() * this.wordList.length);
 return this.wordList[randomIndex].toUpperCase();
}

displayWord() {
 return this.selectedWord
  .split('')
  .map(letter => (this.guesses.includes(letter) ? letter : '_'))
  .join(' ');
}
```

```
makeGuess(letter) {
 letter = letter.toUpperCase();

 if (!this.guesses.includes(letter)) {
  this.guesses.push(letter);
  if (!this.selectedWord.includes(letter)) {
   this.maxAttempts--;

   if (this.maxAttempts === 0) {
    console.log(`Sorry, you're out of attempts. The word was: ${this.selectedWord}`);
    process.exit();
   }
  }

  console.log(`Attempts left: ${this.maxAttempts}`);
  console.log(`Current word: ${this.displayWord()}`);

  if (!this.displayWord().includes('_')) {
   console.log('Congratulations! You guessed the word!');
   process.exit();
  }
 }
}

// Example usage
```

```
const wordList = ['javascript', 'python', 'java', 'ruby', 'typescript'];
const game = new WordGuessingGame(wordList);

console.log('Welcome to the Word Guessing Game!');
console.log(`Attempts left: ${game.maxAttempts}`);
console.log(`Current word: ${game.displayWord()}`);

// Simulate the game loop (you can replace this with actual user input in a real environment)
game.makeGuess('a');
game.makeGuess('e');
game.makeGuess('i');
game.makeGuess('o');
game.makeGuess('u');
game.makeGuess('s');
```

143. Calculate the Euler's Totient Function

This program calculates Euler's Totient Function, also known as Euler's Phi Function, which counts the positive integers up to a given integer `n` that are relatively prime to `n`. The formula for Euler's Totient Function is: ϕ(n) = n * (1 - 1/p1) * (1 - 1/p2) * ... * (1 - 1/pk), where p1, p2, ..., pk are the distinct prime factors of `n`.

```
function eulerTotientFunction(n) {
 if (n <= 0) {
  return "Input must be a positive integer.";
 }

let result = n; // Initialize result as n

// Iterate through all prime factors of n
for (let p = 2; p * p <= n; p++) {
 if (n % p === 0) {
  while (n % p === 0) {
   n /= p;
  }
  result -= result / p; // Apply the formula for prime factors
 }
}

// If n is a prime number greater than 1
```

```
  if (n > 1) {
    result -= result / n;
  }

  return Math.floor(result); // Return the result as an integer
}

// Example usage
const n = 12; // Replace with the positive integer for which you want to calculate Euler's Totient Function
const result = eulerTotientFunction(n);
console.log(`Euler's Totient Function for ${n} is: ${result}`);
```

144. Basic Memory Matching Game

Creating a basic memory matching game involves displaying a grid of face-down cards, allowing the player to flip two cards at a time, and checking for matches.

```
class MemoryMatchingGame {
 constructor(cards) {
  this.cards = cards.concat(cards); // Duplicate the cards to create pairs
  this.shuffleCards();
  this.board = this.createBoard();
  this.flippedCards = [];
  this.attempts = 0;
 this.matches = 0;
}

shuffleCards() {
 for (let i = this.cards.length - 1; i > 0; i--) {
  const j = Math.floor(Math.random() * (i + 1));
  [this.cards[i], this.cards[j]] = [this.cards[j], this.cards[i]];
 }
}

createBoard() {
 const board = [];
```

```
  for (let i = 0; i < 4; i++) {
    const row = [];
    for (let j = 0; j < 4; j++) {
      row.push(this.cards[i * 4 + j]);
    }
    board.push(row);
  }
  return board;
}

displayBoard() {
  for (const row of this.board) {
    console.log(row.map(card => (card.faceUp ? card.value : 'X')).join(' '));
  }
}

flipCard(row, col) {
  const card = this.board[row][col];
  if (!card.faceUp && this.flippedCards.length < 2) {
    card.faceUp = true;
    this.flippedCards.push(card);

    if (this.flippedCards.length === 2) {
      this.attempts++;
      if (this.flippedCards[0].value === this.flippedCards[1].value) {
```

```
    this.matches++;
    console.log('Match found!');
  } else {
    console.log('No match. Try again.');
    this.flippedCards.forEach(card => (card.faceUp = false));
  }

  this.flippedCards = [];
}

this.displayBoard();

if (this.matches === this.cards.length / 2) {
      console.log(`Congratulations! You completed the game in ${this.attempts} attempts.`);
      process.exit();
    }
  }
 }
}

// Example usage
const cards = [
 { value: 'A', faceUp: false },
 { value: 'B', faceUp: false },
 { value: 'C', faceUp: false },
 { value: 'D', faceUp: false },
```

```
  { value: 'A', faceUp: false },
  { value: 'B', faceUp: false },
  { value: 'C', faceUp: false },
  { value: 'D', faceUp: false }
];

const game = new MemoryMatchingGame(cards);

console.log('Welcome to the Memory Matching Game!');
game.displayBoard();

// Simulate the game loop (you can replace this with actual user input in a real environment)
game.flipCard(0, 0);
game.flipCard(1, 1);
game.flipCard(2, 2);
game.flipCard(3, 3);
game.flipCard(0, 1);
game.flipCard(1, 0);
```

145. Check if a Number is a Vampire Number

A vampire number is a composite (non-prime) natural number with an even number of digits, which can be factored into two integers, both having half as many digits as the original number, and where the digits of the original number can be rearranged to form the digits of the two smaller numbers. For example, 1260 is a vampire number because it can be factored into 21 and 60, and the digits of 1260 can be rearranged to form the digits of 21 and 60.

```
function isVampireNumber(number) {
 const strNumber = number.toString();
 const length = strNumber.length;

if (length % 2 !== 0) {
 // Vampire numbers must have an even number of digits
 return false;
}

const pairs = getDigitPairs(strNumber);

for (const [pair1, pair2] of pairs) {
 if (pair1 === 0 || pair2 === 0) {
  // Exclude cases where one of the pairs has a leading zero
  continue;
 }
```

```
    if (number === pair1 * pair2) {
      return true;
    }
  }
 }

 return false;
}

function getDigitPairs(strNumber) {
 const pairs = [];
 const length = strNumber.length;

 for (let i = 0; i < length; i++) {
  for (let j = 0; j < length; j++) {
   if (i !== j) {
    const pair1 = parseInt(strNumber[i] + strNumber[j], 10);
    const pair2 = parseInt(strNumber.split('').filter((_, index) => index !== i && index !== j).join(''), 10);
    pairs.push([pair1, pair2]);
   }
  }
 }

 return pairs;
}

// Example usage
```

```
const vampireNumber = 1260; // Replace with the number you want to check
if (isVampireNumber(vampireNumber)) {
  console.log(`${vampireNumber} is a Vampire Number!  `);
} else {
  console.log(`${vampireNumber} is not a Vampire Number. ?? `);
}
```

146. Basic Simon Says Game

Creating a basic Simon Says game involves generating and displaying a sequence of colors, then challenging the player to repeat the sequence.

```
class SimonSaysGame {
 constructor(colors) {
  this.colors = colors;
  this.sequence = [];
  this.playerSequence = [];
  this.round = 1;
  this.generateSequence();
}

generateSequence() {
 for (let i = 0; i < this.round; i++) {
  const randomIndex = Math.floor(Math.random() * this.colors.length);
  this.sequence.push(this.colors[randomIndex]);
 }
}

displaySequence() {
 console.log(`Round ${this.round} - Simon says:`);
 for (const color of this.sequence) {
```

146. Basic Simon Says Game

Creating a basic Simon Says game involves generating and displaying a sequence of colors, then challenging the player to repeat the sequence.

```
class SimonSaysGame {
 constructor(colors) {
  this.colors = colors;
  this.sequence = [];
  this.playerSequence = [];
  this.round = 1;
  this.generateSequence();
}

generateSequence() {
 for (let i = 0; i < this.round; i++) {
  const randomIndex = Math.floor(Math.random() * this.colors.length);
  this.sequence.push(this.colors[randomIndex]);
 }
}

displaySequence() {
 console.log(`Round ${this.round} - Simon says:`);
 for (const color of this.sequence) {
```

```
    console.log(color);
    // Simulate a delay between colors (you can replace this with actual delays in a real environment)
    this.sleep(1000);
    console.clear();
  }
}

getPlayerInput() {
  console.log('Your turn! Enter the sequence (e.g., "red blue green"):');
  // Simulate user input (you can replace this with actual user input in a real environment)
  const userInput = prompt('Enter the sequence:').toLowerCase();
  this.playerSequence = userInput.split(' ');
}

checkSequence() {
  for (let i = 0; i < this.round; i++) {
    if (this.playerSequence[i] !== this.sequence[i]) {
      console.log('Wrong sequence! Game over.');
      process.exit();
    }
  }
  console.log('Correct sequence! Next round.');
  this.round++;
}
```

```
  sleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
  }

  async play() {
    while (true) {
      this.displaySequence();
      this.getPlayerInput();
      this.checkSequence();
    }
  }
}

// Example usage
const colors = ['red', 'blue', 'green', 'yellow'];
const simonSaysGame = new SimonSaysGame(colors);

console.log('Welcome to Simon Says! Repeat the sequence correctly to advance.');

// Start the game loop
simonSaysGame.play();
```

147. Calculate the Area of a Hexagon

This program calculates the area of a regular hexagon given the side length (`s`). The formula for the area of a regular hexagon is `Area = (3 * √3 * s^2) / 2` , where `s` is the side length.

```
function calculateHexagonArea(sideLength) {
  const area = (3 * Math.sqrt(3) / 2) * Math.pow(sideLength, 2);
  return area;
}

// Example usage
const sideLength = 5; // Replace with the length of a side of your hexagon
const hexagonArea = calculateHexagonArea(sideLength);
console.log(`The area of the hexagon is: ${hexagonArea.toFixed(2)}`);
```

148. Check if a Number is a Pronic Number

This program checks if a given non-negative integer is a pronic number, also known as a rectangular number or oblong number. A pronic number is a product of two consecutive integers. The nth pronic number is given by the formula: `n * (n + 1)`.

```
function isPronicNumber(number) {
 for (let i = 0; i * (i + 1) <= number; i++) {
  if (i * (i + 1) === number) {
   return true;
  }
 }
 return false;
}

// Example usage
const pronicNumber = 6; // Replace with the number you want to check
if (isPronicNumber(pronicNumber)) {
 console.log(`${pronicNumber} is a Pronic Number!`);
} else {
 console.log(`${pronicNumber} is not a Pronic Number.`);
}
```

149. Basic Minesweeper Game

Creating a basic Minesweeper game involves generating a grid with mines and numbers, allowing the player to uncover cells, and providing feedback on the presence of mines.

```
class MinesweeperGame {
 constructor(rows, cols, numMines) {
  this.rows = rows;
  this.cols = cols;
  this.numMines = numMines;
  this.board = this.createBoard();
  this.gameOver = false;
}

createBoard() {
 const board = Array.from({ length: this.rows }, () => Array(this.cols).fill(0));

 // Place mines randomly
 for (let i = 0; i < this.numMines; i++) {
  let row, col;
  do {
   row = Math.floor(Math.random() * this.rows);
   col = Math.floor(Math.random() * this.cols);
  } while (board[row][col] === 'X');
```

```
  board[row][col] = 'X';
}

// Calculate numbers around mines
for (let i = 0; i < this.rows; i++) {
 for (let j = 0; j < this.cols; j++) {
  if (board[i][j] === 'X') {
   continue;
  }

  for (let row = i - 1; row <= i + 1; row++) {
   for (let col = j - 1; col <= j + 1; col++) {
    if (row >= 0 && row < this.rows && col >= 0 && col < this.cols && board[row][col] === 'X') {
     board[i][j]++;
    }
   }
  }
 }
}

 return board;
}

displayBoard() {
 console.log('Minesweeper Board:');
 for (const row of this.board) {
```

```
    console.log(row.map(cell => (cell === 'X' ? 'X' : ' ')).join(' '));
  }
 }

 uncoverCell(row, col) {
  if (this.gameOver || row < 0 || row >= this.rows || col < 0 || col >= this.cols) {
   return;
  }

  const cell = this.board[row][col];
  if (cell === 'X') {
   console.log('Game over! You hit a mine.');
   this.gameOver = true;
  } else if (cell === 0) {
   this.board[row][col] = ' ';
   this.uncoverNeighbors(row, col);
  } else {
   this.board[row][col] = cell.toString();
  }

  this.checkWin();
 }

 uncoverNeighbors(row, col) {
  for (let i = row - 1; i <= row + 1; i++) {
   for (let j = col - 1; j <= col + 1; j++) {
```

```
    if (i >= 0 && i < this.rows && j >= 0 && j < this.cols && this.board[i][j] === 0) {
      this.board[i][j] = ' ';
      this.uncoverNeighbors(i, j);
    } else if (i >= 0 && i < this.rows && j >= 0 && j < this.cols && this.board[i][j] !== 'X') {
      this.board[i][j] = this.board[i][j].toString();
    }
   }
  }
}

checkWin() {
  let uncoveredCount = 0;
  for (const row of this.board) {
    uncoveredCount += row.filter(cell => cell !== 'X').length;
   }

   if (uncoveredCount === this.rows * this.cols - this.numMines) {
    console.log('Congratulations! You win!');
    this.gameOver = true;
   }
  }
}

// Example usage
const minesweeperGame = new MinesweeperGame(5, 5, 3);
```

```
minesweeperGame.displayBoard();

// Simulate the game loop (you can replace this with actual user input in a real environment)
minesweeperGame.uncoverCell(0, 0);
minesweeperGame.uncoverCell(2, 2);
minesweeperGame.uncoverCell(1, 1);
minesweeperGame.uncoverCell(3, 3);
minesweeperGame.uncoverCell(4, 4);
```

150. Find the Area of a Pentagon

This program calculates the area of a regular pentagon given the side length (`s`). The formula for the area of a regular pentagon is `Area = (1/4) * √(5 * (5 + 2 * √5)) * s^2` , where `s` is the side length.

```
function calculatePentagonArea(sideLength) {
 const area = (1 / 4) * Math.sqrt(5 * (5 + 2 * Math.sqrt(5))) * Math.pow(sideLength, 2);
 return area;
}

// Example usage
const sideLength = 4; // Replace with the length of a side of your pentagon
const pentagonArea = calculatePentagonArea(sideLength);
console.log(`The area of the pentagon is: ${pentagonArea.toFixed(2)}`);
```

151. Check if a Number is a Cube Number

This program checks if a given non-negative integer is a cube number. A cube number is a number that can be expressed as the cube of an integer. The nth cube number is given by the formula: `n^3`.

```
function isCubeNumber(number) {
 const cubeRoot = Math.round(Math.pow(number, 1 / 3));
 return Math.pow(cubeRoot, 3) === number;
}

// Example usage
const numberToCheck = 27; // Replace with the number you want to check
if (isCubeNumber(numberToCheck)) {
 console.log(`${numberToCheck} is a Cube Number! ✅`);
} else {
 console.log(`${numberToCheck} is not a Cube Number. ❌`);
}
```

152. Basic Typing Speed Test

Creating a basic typing speed test involves generating a random string, measuring the time it takes for the user to type the string, and calculating the typing speed.

```
class TypingSpeedTest {
 constructor() {
  this.words = [
   'programming', 'javascript', 'developer', 'challenge', 'keyboard',
   'speed', 'test', 'accuracy', 'practice', 'learning'
  ];
  this.text = this.generateRandomText();
  this.startTime = 0;
  this.endTime = 0;
  this.userInput = '';
}

generateRandomText() {
  const randomIndex = Math.floor(Math.random() * this.words.length);
  return this.words[randomIndex];
}

startTest() {
  this.text = this.generateRandomText();
  this.startTime = Date.now();
```

```
  console.log(`Type the following text:\n"${this.text}"`);
 }

 endTest() {
  this.endTime = Date.now();
  const elapsedTime = (this.endTime - this.startTime) / 1000; // Convert to seconds
  const wordsPerMinute = (this.text.split(' ').length / elapsedTime) * 60;
  console.log(`You typed at a speed of ${wordsPerMinute.toFixed(2)} words per minute.`);
 }

 getUserInput(input) {
  this.userInput = input.trim();
  if (this.userInput === this.text) {
    console.log('Congratulations! You typed the correct text.');
    this.endTest();
   } else {
    console.log('Incorrect. Try again.');
   }
  }
}

// Example usage
const typingTest = new TypingSpeedTest();
typingTest.startTest();

// Simulate user input (you can replace this with actual user input in a real environment)
```

```
const userInput = 'javascript developer';
typingTest.getUserInput(userInput);
```

153. Find the Area of a Cube

This program calculates the surface area of a cube given the length of its side (`s`). The formula for the surface area of a cube is `Surface Area = 6 * s^2` , where `s` is the length of a side.

```
function calculateCubeSurfaceArea(sideLength) {
 const surfaceArea = 6 * Math.pow(sideLength, 2);
 return surfaceArea;
}

// Example usage
const sideLength = 4; // Replace with the length of a side of your cube
const cubeSurfaceArea = calculateCubeSurfaceArea(sideLength);
console.log(`The surface area of the cube is: ${cubeSurfaceArea}`);
```

154. Find the Area of a Cone

This program calculates the surface area of a cone given the radius of its base (`r`) and the slant height (`l`). The formula for the surface area of a cone is **`Surface Area = π * r * (r + l)`**, where `π` is the mathematical constant Pi.

```
function calculateCubeSurfaceArea(sideLength) {
 const surfaceArea = 6 * Math.pow(sideLength, 2);
 return surfaceArea;
}

// Example usage
const sideLength = 4; // Replace with the length of a side of your cube
const cubeSurfaceArea = calculateCubeSurfaceArea(sideLength);
console.log(`The surface area of the cube is: ${cubeSurfaceArea}`);
```

155. Check if a Number is a Happy Number

A happy number is a positive integer that, when repeatedly replaced by the sum of the squares of its digits, eventually reaches the number 1. If a number becomes 1, it is a happy number; otherwise, it falls into a cycle that does not include 1.

```
function isHappyNumber(number) {
 const seenNumbers = new Set();

 while (number !== 1 && !seenNumbers.has(number)) {
  seenNumbers.add(number);
  number = sumOfSquaredDigits(number);
 }

 return number === 1;
}

function sumOfSquaredDigits(number) {
 return Array.from(String(number), Number)
  .reduce((sum, digit) => sum + Math.pow(digit, 2), 0);
}

// Example usage
const happyNumber = 19; // Replace with the number you want to check
if (isHappyNumber(happyNumber)) {
 console.log(`${happyNumber} is a Happy Number! ��`);
```

```
} else {
  console.log(`${happyNumber} is not a Happy Number. ☹ `);
}
```

156. Calculate the Area of a Triangular Prism

This program calculates the surface area of a triangular prism given the lengths of its base sides (`a` , `b` , and `c`) and the height of the prism (`h`). The formula for the surface area of a triangular prism is `Surface Area = (Perimeter of the base * Height) + (2 * Area of the base)` , where the perimeter of the base is `Perimeter = a + b + c` and the area of the base is calculated using Heron's formula.

```
function calculateTriangularPrismSurfaceArea(a, b, c, height) {
  const baseArea = 0.25 * Math.sqrt((-a + b + c) * (a - b + c) * (a + b - c) * (a + b + c));
  const perimeter = a + b + c;
  const surfaceArea = baseArea + perimeter * height;
  return surfaceArea;
}

// Example usage
const sideA = 3;  // Replace with the length of side A of the triangular base
const sideB = 4;  // Replace with the length of side B of the triangular base
const sideC = 5;  // Replace with the length of side C of the triangular base
const prismHeight = 6;  // Replace with the height of the triangular prism
const triangularPrismSurfaceArea = calculateTriangularPrismSurfaceArea(sideA, sideB, sideC, prismHeight);
console.log(`The surface area of the triangular prism is: ${triangularPrismSurfaceArea.toFixed(2)}`);
```

157. Check if a Number is a Fibonacci Prime

This program checks if a given positive integer is both a Fibonacci number and a prime number. A Fibonacci prime is a number that is both a Fibonacci number and a prime number.

```
function isPrime(number) {
 if (number <= 1) {
  return false;
 }
 for (let i = 2; i <= Math.sqrt(number); i++) {
  if (number % i === 0) {
   return false;
  }
 }
 return true;
}
```

```
function isPerfectSquare(number) {
  const sqrt = Math.sqrt(number);
  return sqrt === Math.floor(sqrt);
}

function isFibonacciPrime(number) {
  return isPrime(number) && isPerfectSquare(5 * Math.pow(number, 2) + 4) || isPerfectSquare(5 * Math.pow(number, 2)
- 4);
}

// Example usage
const fibonacciPrimeCandidate = 13; // Replace with the number you want to check
if (isFibonacciPrime(fibonacciPrimeCandidate)) {
  console.log(`${fibonacciPrimeCandidate} is a Fibonacci Prime!`);
} else {
  console.log(`${fibonacciPrimeCandidate} is not a Fibonacci Prime.`);
}
```

158. Check if a Number is a Squareful Number

This program checks if a given positive integer is a squareful number. A squareful number is a positive integer where the square of each prime factor occurs in its prime factorization.

```
function isSquarefulNumber(number) {
 const factors = getFactors(number);
 const uniqueFactors = [...new Set(factors)];

 for (const factor of uniqueFactors) {
  const count = factors.filter((num) => num === factor).length;
  if (count % 2 !== 0) {
   return false;
  }
 }

 return true;
}

function getFactors(number) {
 const factors = [];
 for (let i = 1; i <= Math.sqrt(number); i++) {
  if (number % i === 0) {
   factors.push(i);
   if (i !== Math.sqrt(number)) {
```

```
      factors.push(number / i);
    }
   }
  }
  return factors;
}

// Example usage
const squarefulNumber = 36; // Replace with the number you want to check
if (isSquarefulNumber(squarefulNumber)) {
  console.log(`${squarefulNumber} is a Squareful Number!`);
} else {
  console.log(`${squarefulNumber} is not a Squareful Number.`);
}
```

159. Find the Area of a Tetrahedron

This program calculates the surface area of a regular tetrahedron given the length of its edges (`a`). The formula for the surface area of a regular tetrahedron is `Surface Area = √3 * a^ 2` , where `a` is the length of an edge.

```
function calculateTetrahedronSurfaceArea(edgeLength) {
  const surfaceArea = Math.sqrt(3) * Math.pow(edgeLength, 2);
  return surfaceArea;
}
```

```
// Example usage
const edgeLength = 5; // Replace with the length of an edge of the tetrahedron
const tetrahedronSurfaceArea = calculateTetrahedronSurfaceArea(edgeLength);
console.log(`The surface area of the tetrahedron is: ${tetrahedronSurfaceArea.toFixed(2)}`);
```

160. Check if a Number is an Automorphic Number

This program checks if a given positive integer is an automorphic number. An automorphic number is a number whose square ends with the number itself. For example, 5 is an automorphic number because 5^2 = 25, and 25 ends with 5.

```
function isAutomorphicNumber(number) {
  const square = number * number;
  const numberDigits = String(number);
  const squareDigits = String(square);

  return squareDigits.endsWith(numberDigits);
}

// Example usage
const automorphicNumber = 25; // Replace with the number you want to check
if (isAutomorphicNumber(automorphicNumber)) {
  console.log(`${automorphicNumber} is an Automorphic Number!`);
} else {
  console.log(`${automorphicNumber} is not an Automorphic Number.`);
}
```

161. Calculate the Area of a Pyramid

This program calculates the surface area of a square pyramid given the length of the base side (`s`) and the slant height (`l`). The formula for the surface area of a square pyramid is `Surface Area = s^2 + 2 * s * l` , where `s` is the length of a base side and `l` is the slant height.

```
function calculatePyramidSurfaceArea(sideLength, slantHeight) {
  const baseArea = Math.pow(sideLength, 2);
  const lateralArea = 2 * sideLength * slantHeight;
  const surfaceArea = baseArea + lateralArea;
  return surfaceArea;
}

// Example usage
const sideLength = 4; // Replace with the length of a side of the pyramid's base
const slantHeight = 5; // Replace with the slant height of the pyramid
const pyramidSurfaceArea = calculatePyramidSurfaceArea(sideLength, slantHeight);
console.log(`The surface area of the pyramid is: ${pyramidSurfaceArea.toFixed(2)}`);
```

162. Check if a Number is a Smith–Morra Gambit Number

This program checks if a given positive integer is a Smith–Morra Gambit number. A Smith–Morra Gambit number is a positive integer that can be expressed as the sum of the digits of its prime factors, including repetitions.

```
function isSmithMorraGambitNumber(number) {
 // Function to calculate the sum of digits
 const digitSum = (n) => n.toString().split('').reduce((acc, digit) => acc + parseInt(digit), 0);

 // Function to calculate the prime factorization of a number
 const primeFactorization = (n) => {
  const factors = [];
  for (let i = 2; i <= n; i++) {
   while (n % i === 0) {
    factors.push(i);
    n /= i;
   }
  }
  return factors;
};

// Get the prime factorization of the number and calculate the sum of its digits
const factors = primeFactorization(number);
const sumOfDigits = factors.reduce((acc, factor) => acc + digitSum(factor), 0);
```

```
  // Check if the sum of digits of prime factors equals the original number
  return sumOfDigits === digitSum(number);
}

// Example: Check if 22 is a Smith–Morra Gambit Number
console.log(isSmithMorraGambitNumber(22)); // Output: true
```

163. Check if a Number is a Solitary Number

This program checks if a number is a solitary number:

```
function getProperDivisorsSum(number) {
  let sum = 1; // Start with 1 as every number is divisible by 1

  for (let i = 2; i <= Math.sqrt(number); i++) {
    if (number % i === 0) {
      sum += i;

      if (i !== number / i) {
        sum += number / i;
      }
    }
  }
```

```
  return sum;
}

function isSolitaryNumber(number) {
  const sum = getProperDivisorsSum(number);
  return number !== sum;
}

// Example usage
const solitaryNumber = 28; // Replace with the number you want to check
if (isSolitaryNumber(solitaryNumber)) {
  console.log(`${solitaryNumber} is a Solitary Number!`);
} else {
  console.log(`${solitaryNumber} is not a Solitary Number.`);
}
```

164. Basic Tower of Hanoi Puzzle

The Tower of Hanoi is a classic problem in computer science and mathematics. It consists of three pegs and a number of disks of different sizes, which can slide onto any peg. The puzzle starts with the disks in a neat stack in ascending order of size on one peg, the smallest at the top. The objective is to move the entire stack to another peg, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty peg.
3. No disk may be placed on top of a smaller disk.

```
function towerOfHanoi(n, source, auxiliary, target) {
 if (n === 1) {
  console.log(`Move disk 1 from ${source} to ${target}`);
  return;
 }

 towerOfHanoi(n - 1, source, target, auxiliary);
 console.log(`Move disk ${n} from ${source} to ${target}`);
 towerOfHanoi(n - 1, auxiliary, source, target);
}

// Example usage
const numberOfDisks = 3;
```

```
const sourcePeg = 'A';
const auxiliaryPeg = 'B';
const targetPeg = 'C';

console.log(`Tower of Hanoi solution for ${numberOfDisks} disks:`);
towerOfHanoi(numberOfDisks, sourcePeg, auxiliaryPeg, targetPeg);

}
```

165. Calculate the Area of a Frustum

This program calculates the surface area of a frustum of a cone given the radii of its top and bottom bases (`r1` and `r2`) and the slant height (`l`). The formula for the surface area of a frustum of a cone is `Surface Area = π * (r1 + r2) * l + π * r1^2 + π * r2^2` , where `π` is the mathematical constant Pi.

```
function surfaceAreaOfFrustum(r1, r2, l) {
  const surfaceArea = Math.PI * (r1 + r2) * l + Math.PI * r1 ** 2 + Math.PI * r2 ** 2;
  console.log(`Surface Area of the frustum is: ${surfaceArea}`);
}

// Example: Calculate the surface area of a frustum with top radius 4, bottom radius 8, and slant height 6
surfaceAreaOfFrustum(4, 8, 6);
```

166. Check if a Number is a Motzkin Number

This program checks if a given positive integer is a Motzkin number. A Motzkin number is a number in

the sequence of Motzkin numbers, which counts the number of different ways of drawing non-intersecting chords between points on a circle (not necessarily touching every point with a chord).

```
function isMotzkinNumber(number) {
 if (number < 0) {
  return false;
 }

 // Use dynamic programming to calculate Motzkin numbers
 const motzkinNumbers = [1, 1];
 for (let n = 2; n <= number; n++) {
  const nextMotzkin = ((2 * n + 1) * motzkinNumbers[n - 1] + (3 * n - 3) * motzkinNumbers[n - 2]) / (n + 2);
  motzkinNumbers.push(nextMotzkin);
 }

 return motzkinNumbers.includes(number);
}

// Example usage
const motzkinNumber = 5; // Replace with the number you want to check
if (isMotzkinNumber(motzkinNumber)) {
 console.log(`${motzkinNumber} is a Motzkin Number!`);
} else {
 console.log(`${motzkinNumber} is not a Motzkin Number.`);
}
```

167. Check if a Number is a Padovan Number

This program checks if a given positive integer is a Padovan number. A Padovan number is a number in the Padovan sequence, which is similar to the Fibonacci sequence but uses a recurrence relation of P(n) = P(n-2) + P(n-3) with initial values P(0) = P(1) = P(2) = 1.

```
function isPadovanNumber(number) {
 if (number < 0) {
  return false;
 }

 const padovanNumbers = [1, 1, 1];
 while (padovanNumbers[padovanNumbers.length - 1] <= number) {
  const nextPadovan = padovanNumbers[padovanNumbers.length - 2] + padovanNumbers[padovanNumbers.length - 3];
  padovanNumbers.push(nextPadovan);
 }

 return padovanNumbers.includes(number);
}

// Example usage
const padovanNumber = 9; // Replace with the number you want to check
if (isPadovanNumber(padovanNumber)) {
 console.log(`${padovanNumber} is a Padovan Number!`);
```

```
} else {
  console.log(`${padovanNumber} is not a Padovan Number.`);
}
```

168. Calculate the Area of a Koch Snowflake

The Koch snowflake is a mathematical curve and one of the earliest fractal curves to have been described. It can be constructed by starting with an equilateral triangle and repeatedly adding smaller equilateral triangles to each side of the existing triangles.

```
function calculateKochSnowflakeArea(sideLength, iterations) {
  const sqrt3 = Math.sqrt(3);
  const area = (4 * sqrt3 / 5) * Math.pow((sideLength / 3), 2) * Math.pow(3 / 2, iterations);
  return area;
}

// Example usage
const sideLength = 100; // Replace with the side length of the original equilateral triangle
const iterations = 4;  // Replace with the number of iterations
const kochSnowflakeArea = calculateKochSnowflakeArea(sideLength, iterations);
console.log(`The area of the Koch snowflake is: ${kochSnowflakeArea.toFixed(2)}`);
```

169. Basic Chatbot

This program is a simple and interactive script that allows users to engage in a text-based conversation with a virtual assistant.

```
function chatbot(message) {
 // Simple rule-based responses
 if (message.toLowerCase().includes('hello')) {
  return 'Hello! How can I help you?';
 } else if (message.toLowerCase().includes('how are you')) {
  return 'I am just a computer program, but thanks for asking!';
 } else if (message.toLowerCase().includes('bye')) {
  return 'Goodbye!';
 } else {
  return 'I didn\'t understand that. Can you please rephrase?';
 }
}

function startChat() {
 const userInput = prompt('You: ');
 if (userInput !== null) {
  const response = chatbot(userInput);
  console.log('Chatbot:', response);
  startChat(); // Continue the conversation
 } else {
```

```
    console.log('Goodbye!');
  }
}

console.log('Chatbot: Hello! How can I help you today?');
startChat();
```

170. RGB to HEX Converter

This program allows you to convert RBG values to hexadecimal.

```
function convertToHex(red, green, blue) {
 const toHex = value => {
  const hex = parseInt(value, 10).toString(16);
  return hex.length === 1 ? '0' + hex : hex;
 };

 const hexRed = toHex(red);
 const hexGreen = toHex(green);
 const hexBlue = toHex(blue);

 return `#${hexRed}${hexGreen}${hexBlue}`;
}

// Example usage
const redValue = prompt('Enter the Red value (0-255):');
const greenValue = prompt('Enter the Green value (0-255):');
const blueValue = prompt('Enter the Blue value (0-255):');

const hexResult = convertToHex(redValue, greenValue, blueValue);
console.log(`HEX: ${hexResult}`);
```

171. Counter

This program allows you to increment, decrement and reset a counter.

```
let count = 0;

function incrementCounter() {
 count++;
 console.log(`Counter: ${count}`);
}

function decrementCounter() {
 count--;
 console.log(`Counter: ${count}`);
}

function resetCounter() {
 count = 0;
 console.log("Counter reset to 0");
}

// Example usage
incrementCounter();
incrementCounter();
decrementCounter();
resetCounter();
```

172. Stop The Bomb

This program to allow you stop an active bomb.

```
let bombTimer;
let countdownInterval;
let timeLeft = 10; // Initial time in seconds

function startBomb() {
  console.log("The bomb is ticking! Hurry, defuse it!");
  displayTimeLeft();
  bombTimer = setTimeout(() => {
    console.log("BOOM! The bomb exploded. Game over!");
    clearInterval(countdownInterval);
  }, timeLeft * 1000);
}

function defuseBomb() {
  clearTimeout(bombTimer);
  console.log("Whew! You successfully defused the bomb!");
  clearInterval(countdownInterval);
}

function displayTimeLeft() {
  countdownInterval = setInterval(() => {
```

```
    console.clear();
    console.log(`Time remaining: ${timeLeft} seconds`);
    timeLeft--;

    if (timeLeft < 0) {
      clearTimeout(bombTimer);
      console.log("BOOM! The bomb exploded. Game over!");
      clearInterval(countdownInterval);
    }
  }, 1000);
}

// Example usage
startBomb();
// Uncomment the line below to simulate successfully defusing the bomb
// defuseBomb();
```

173. IP Address Generator

Generating IP addresses programmatically can be done using a programming language like JavaScript. Below is a simple example of an IP address generator in JavaScript. Note that this example generates random IP addresses for educational purposes and should not be used for any malicious activities.

```
function generateRandomIp() {
  const octet = () => Math.floor(Math.random() * 256);
  return `${octet()}.${octet()}.${octet()}.${octet()}`;
```

```
}

function generateMultipleRandomIps(count) {
  const ipAddresses = [];
  for (let i = 0; i < count; i++) {
    ipAddresses.push(generateRandomIp());
  }
  return ipAddresses;
}

// Example: Generate and print 5 random IP addresses
const numberOfIps = 5;
const randomIps = generateMultipleRandomIps(numberOfIps);
console.log('Random IP Addresses:', randomIps);
```

174. Digital clock

Deploy a digital clock with this awesome program.

```
function updateClock() {
  const now = new Date();
  const hours = now.getHours();
  const minutes = now.getMinutes();
  const seconds = now.getSeconds();

  const formattedTime = `${hours}:${minutes}:${seconds}`;

  console.clear(); // Clears the console to update the time
  console.log("Current Time: " + formattedTime);
}

// Update the clock every second
setInterval(updateClock, 1000);
```

175. Random Number Generator

This program generates a random number between 1 and 10 and displays it in the console.

```
function generateRandomNumber(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}

const randomNumber = generateRandomNumber(1, 10);
console.log("Random Number:", randomNumber);
```

176. Simple JavaScript Quiz

This program presents a simple JavaScript quiz with multiple-choice questions. It asks the user for answers and provides feedback in the console.

```
function runQuiz() {
 const questions = [
  {
   question: "What is the capital of France?",
   options: ["Berlin", "Madrid", "Paris", "Rome"],
   correctAnswer: "Paris"
  },
  {
   question: "Which programming language is this quiz written in?",
   options: ["JavaScript", "Python", "Java", "C++"],
   correctAnswer: "JavaScript"
  },
  {
   question: "What is the largest mammal?",
   options: ["Elephant", "Whale Shark", "Blue Whale", "Giraffe"],
   correctAnswer: "Blue Whale"
  }
];

let score = 0;
```

```
for (const question of questions) {
 console.log(question.question);
 for (let i = 0; i < question.options.length; i++) {
  console.log(` ${i + 1}. ${question.options[i]} `);
 }

 const userAnswerIndex = prompt("Enter the number of your answer:");
 const userAnswer = question.options[userAnswerIndex - 1];

 if (userAnswer === question.correctAnswer) {
  console.log("Correct!\n");
  score++;
 } else {
  console.log(` Wrong! The correct answer is ${question.correctAnswer}\n `);
 }
}

 console.log(` Your final score: ${score} out of ${questions.length} `);
}

// Run the quiz
runQuiz();
```

177. Text-based RPG Adventure

This program takes the user on a simple text-based RPG adventure where they make choices to navigate through a story.

```
function textRPGAdventure() {
 console.log("Welcome to the Text-based RPG Adventure!");
 console.log("You find yourself in a dark cave. What do you want to do?");

 const choices = [
  { option: "Explore the cave", outcome: "You discover a hidden treasure!", next: "treasure" },
  { option: "Go deeper into the cave", outcome: "Uh-oh! A dragon appears! You narrowly escape.", next: "dragon" },
  { option: "Exit the cave", outcome: "You emerge into the sunlight. Adventure awaits!", next: "end" }
 ];

 let currentChoice = 0;

 while (currentChoice < choices.length) {
  const userChoice = prompt(choices[currentChoice].option);
  console.log(choices[currentChoice].outcome);

  if (userChoice.toLowerCase() === "exit") {
   console.log("You exit the adventure. Goodbye!");
   break;
  }
```

```
    currentChoice = choices.findIndex(choice => choice.next === userChoice.toLowerCase());
    if (currentChoice === -1) {
      console.log("Invalid choice. Try again or type 'exit' to end the adventure.");
    }
  }
}

// Start the text-based RPG adventure
textRPGAdventure();
```

178. Dice Rolling Simulator

This program simulates rolling a six-sided die, providing a random result between 1 and 6.

```
function rollDice() {
  const result = Math.floor(Math.random() * 6) + 1;
  console.log(`You rolled a ${result}`);
}

// Simulate rolling the dice
rollDice();
```

179. Seconds to Time Converter

This program takes a number of seconds as input and converts it into hours, minutes, and seconds, displaying the result in the console.

```
function convertSecondsToTime(seconds) {
  const hours = Math.floor(seconds / 3600);
  const minutes = Math.floor((seconds % 3600) / 60);
  const remainingSeconds = seconds % 60;

  console.log(`Time: ${hours} hours, ${minutes} minutes, ${remainingSeconds} seconds`);
}

// Example: Convert 3665 seconds to time
const inputSeconds = 3665;
convertSecondsToTime(inputSeconds);
```

180. Bar Chart Generator

This program takes an array of numerical values and creates a simple horizontal bar chart in the console.

```
function generateBarChart(data) {
 const maxValue = Math.max(...data);

 for (let value of data) {
  const barLength = Math.round((value / maxValue) * 20);
  const bar = '■'.repeat(barLength).padEnd(20, ' ');

  console.log(`${value} | ${bar}`);
 }
}

// Example: Generate a bar chart for data
const chartData = [5, 8, 12, 4, 6];
generateBarChart(chartData);
```

181. Right-Angled Triangle Pattern

This program generates a right-angled triangle pattern using asterisks and displays it in the console.

```
function generateRightAngledTriangle(height) {
 for (let i = 1; i <= height; i++) {
  const stars = '*'.repeat(i);
  console.log(stars);
 }
}

// Example: Generate a right-angled triangle with a height of 5
const triangleHeight = 5;
generateRightAngledTriangle(triangleHeight);
```

182. Text-Based Calculator

This program allows the user to input mathematical expressions (e.g., "2 + 3 * 4") and evaluates and displays the result in the console.

```
function textBasedCalculator(expression) {
 try {
  const result = eval(expression);
  console.log(`Result: ${result}`);
 } catch (error) {
  console.error('Error:', error.message);
 }
}

// Example: Evaluate a mathematical expression
const userInput = prompt('Enter a mathematical expression:');
textBasedCalculator(userInput);
```

183. Number Pyramid Generator

This program generates a number pyramid pattern and displays it in the console.

```
function generateNumberPyramid(height) {
 for (let i = 1; i <= height; i++) {
  const spaces = ' '.repeat(height - i);
  const numbers = Array.from({ length: i }, (_, index) => index + 1).join(' ');
  console.log(spaces + numbers);
 }
}

// Example: Generate a number pyramid with a height of 4
const pyramidHeight = 4;
generateNumberPyramid(pyramidHeight);
```

184. Diamond Pattern Generator

This program generates a diamond pattern using characters and displays it in the console.

```
function generateDiamondPattern(height) {
 const midpoint = Math.ceil(height / 2);

 for (let i = 1; i <= height; i++) {
  const spaces = Math.abs(midpoint - i);
  const stars = '*'.repeat(height - 2 * spaces);
  console.log(' '.repeat(spaces) + stars);
 }
}

// Example: Generate a diamond pattern with a height of 5
const diamondHeight = 5;
generateDiamondPattern(diamondHeight);
```

185. Hangman Game

This program simulates a text-based Hangman game. The player tries to guess a word by suggesting letters, and the program displays the progress and remaining attempts.

```
function hangmanGame() {
 const words = ["javascript", "hangman", "programming", "developer", "challenge"];
 const selectedWord = words[Math.floor(Math.random() * words.length)];
 let guessedWord = "_".repeat(selectedWord.length);
 let remainingAttempts = 6;
 const guessedLetters = [];

 console.log("Welcome to the Hangman Game!");
console.log("Try to guess the word.");

while (remainingAttempts > 0) {
 console.log(`Word: ${guessedWord}`);
 console.log(`Guessed Letters: ${guessedLetters.join(", ")}`);
 console.log(`Attempts Remaining: ${remainingAttempts}`);

 const userGuess = prompt("Enter a letter:").toLowerCase();

 if (!userGuess || !userGuess.match(/[a-z]/) || userGuess.length !== 1) {
  console.log("Invalid input. Please enter a single letter.");
  continue;
 }
```

```
if (guessedLetters.includes(userGuess)) {
 console.log("You already guessed that letter. Try again.");
 continue;
}

guessedLetters.push(userGuess);

if (selectedWord.includes(userGuess)) {
 for (let i = 0; i < selectedWord.length; i++) {
  if (selectedWord[i] === userGuess) {
   guessedWord = guessedWord.substring(0, i) + userGuess + guessedWord.substring(i + 1);
  }
 }

 if (!guessedWord.includes("_")) {
  console.log(`Congratulations! You guessed the word: ${selectedWord}`);
  break;
 }
} else {
 console.log("Incorrect guess. Try again.");
 remainingAttempts--;

 if (remainingAttempts === 0) {
  console.log(`Out of attempts. The word was: ${selectedWord}`);
 }
```

```
    }
  }
}

// Start the Hangman game
hangmanGame();
```

186. Countdown Timer

This program allows the user to set a countdown time in seconds. The timer counts down, and the program displays the remaining time.

```
function countdownTimer() {
 const userTime = prompt("Enter the countdown time in seconds:");
 const countdownTime = parseInt(userTime, 10);

 if (isNaN(countdownTime) || countdownTime <= 0) {
  console.log("Invalid input. Please enter a positive number.");
  return;
 }

 console.log(`Countdown started for ${countdownTime} seconds.`);

 let secondsRemaining = countdownTime;

 const intervalId = setInterval(() => {
  console.clear();
  console.log(`Time Remaining: ${secondsRemaining} seconds`);

  if (secondsRemaining === 0) {
   console.log("Countdown completed!");
   clearInterval(intervalId);
  } else {
```

```
    secondsRemaining--;
   }
  }, 1000);
}

// Start the countdown timer
countdownTimer();
```

187. Mad Libs Game

This program prompts the user for various types of words (e.g., nouns, verbs) and constructs a humorous story using those inputs.

```
function madLibsGame() {
  console.log("Welcome to the Mad Libs Game!");
  console.log("Let's create a funny story.");

  const noun = prompt("Enter a noun:");
  const verb = prompt("Enter a verb:");
  const adjective = prompt("Enter an adjective:");
  const adverb = prompt("Enter an adverb:");
  const animal = prompt("Enter an animal:");
  const sound = prompt("Enter a sound:");

  console.clear();
  console.log("Here's your Mad Libs story:");

  console.log(`Once upon a time, there was a ${adjective} ${noun} that loved to ${verb} ${adverb}.`);
  console.log(`One day, it met a ${animal} that made a ${sound} noise.`);
  console.log("Everyone in the neighborhood heard the noise and started to join the fun!");
  console.log("And they all lived happily ever after.");

  console.log("The end.");
}
```

```
// Start the Mad Libs game
madLibsGame();
```

188. Guess the Number Game

This program generates a random number, and the player has to guess it within a certain number of attempts. The game provides feedback on whether the guess is too high, too low, or correct.

```
function guessTheNumberGame() {
 const maxNumber = 100;
 const maxAttempts = 10;
 const secretNumber = Math.floor(Math.random() * maxNumber) + 1;

 console.log("Welcome to the Guess the Number Game!");
 console.log(`I've picked a number between 1 and ${maxNumber}. Try to guess it!`);

 for (let attempts = 1; attempts <= maxAttempts; attempts++) {
  const userGuess = parseInt(prompt("Enter your guess:"), 10);

  if (isNaN(userGuess) || userGuess < 1 || userGuess > maxNumber) {
   console.log(`Invalid guess. Please enter a number between 1 and ${maxNumber}.`);
   attempts--;
   continue;
  }

  if (userGuess === secretNumber) {
   console.log(`Congratulations! You guessed the number in ${attempts} attempts.`);
   return;
  } else if (userGuess < secretNumber) {
```

```
    console.log("Too low. Try again!");
   } else {
    console.log("Too high. Try again!");
   }
  }

  console.log(`Sorry, you've run out of attempts. The number was ${secretNumber}.`);
}
// Start the Guess the Number Game
guessTheNumberGame();
```

189. Whack-a-Mole Game

This program simulates a text-based Whack-a-Mole game. The player must "whack" the mole as it randomly appears in different holes.

```
function whackAMoleGame() {
  const totalHoles = 5;
  const molesToWhack = 3;
  const moleSymbol = "M";
  const emptyHoleSymbol = "_";

  let score = 0;
  let remainingAttempts = 10;

  console.log("Welcome to the Whack-a-Mole Game!");
```

```
console.log(`Try to whack ${molesToWhack} moles within ${remainingAttempts} attempts.`);

while (remainingAttempts > 0 && molesToWhack > 0) {
 const holes = Array.from({ length: totalHoles }, () => emptyHoleSymbol);
 const randomMoleIndex = Math.floor(Math.random() * totalHoles);
 holes[randomMoleIndex] = moleSymbol;

 console.log(`\nHoles: ${holes.join(" ")}`);

 const userWhackIndex = parseInt(prompt("Enter the index to whack the mole:"), 10);

 if (isNaN(userWhackIndex) || userWhackIndex < 0 || userWhackIndex >= totalHoles) {
  console.log(`Invalid input. Please enter a number between 0 and ${totalHoles - 1}.`);
  continue;
 }

 if (holes[userWhackIndex] === moleSymbol) {
  console.log("Whack! You got the mole!");
  score++;
  molesToWhack--;
 } else {
  console.log("Missed! No mole at that hole.");
 }

 remainingAttempts--;
 console.log(`Score: ${score} | Attempts remaining: ${remainingAttempts}`);
}
```

```
  if (molesToWhack === 0) {
    console.log("Congratulations! You whacked all the moles!");
  } else {
    console.log("Game over. You ran out of attempts.");
  }
}

// Start the Whack-a-Mole Game
whackAMoleGame();
```

190. Coin Tossing Game

This program simulates a coin tossing game. The player guesses whether the next coin toss will result in heads or tails.

```
function coinTossingGame() {
 const coinSides = ["Heads", "Tails"];
 let correctGuesses = 0;
 let totalGuesses = 0;

 console.log("Welcome to the Coin Tossing Game!");

 while (true) {
const playerGuess = prompt("Heads or Tails?").toLowerCase();

if (!["heads", "tails"].includes(playerGuess)) {
 console.log("Invalid input. Please enter 'Heads' or 'Tails'.");
 continue;
}

const coinResult = coinSides[Math.floor(Math.random() * 2)];

console.log(`Coin toss result: ${coinResult}`);

if (playerGuess === coinResult.toLowerCase()) {
 console.log("Correct guess! Well done!");
 correctGuesses++;
```

```
    } else {
      console.log("Wrong guess. Better luck next time!");
    }

    totalGuesses++;

    const playAgain = confirm("Do you want to toss the coin again?");
    if (!playAgain) {
      console.log(`Game over. You made ${correctGuesses} correct guesses out of ${totalGuesses} total guesses.`);
      break;
    }
  }
}

// Start the Coin Tossing Game
coinTossingGame();
```

191. Fibonacci Number Generator

This program generates and displays Fibonacci numbers up to a specified limit.

```
function fibonacciNumberGenerator() {
 const limit = parseInt(prompt("Enter the limit for Fibonacci numbers:"), 10);

 if (isNaN(limit) || limit < 1) {
  console.log("Invalid input. Please enter a positive number.");
  return;
 }

 let fibonacciNumbers = [0, 1];

 for (let i = 2; i <= limit; i++) {
  fibonacciNumbers[i] = fibonacciNumbers[i - 1] + fibonacciNumbers[i - 2];
 }

 console.log(`Fibonacci numbers up to ${limit}:`);
 console.log(fibonacciNumbers.slice(0, limit + 1));
}

// Start the Fibonacci Number Generator
fibonacciNumberGenerator();
```

192. Math Quiz Game

This program simulates a text-based Math Quiz game. The player is asked random addition questions and scores points for correct answers.

```
function mathQuizGame() {
 const maxQuestions = 5;
 let score = 0;

 console.log("Welcome to the Math Quiz Game!");
 console.log("Answer the following addition questions:");

 for (let i = 0; i < maxQuestions; i++) {
  const num1 = Math.floor(Math.random() * 10) + 1;
  const num2 = Math.floor(Math.random() * 10) + 1;
  const correctAnswer = num1 + num2;

  const userAnswer = parseInt(prompt(`Question ${i + 1}: ${num1} + ${num2} = ?`), 10);

  if (!isNaN(userAnswer) && userAnswer === correctAnswer) {
   console.log("Correct! Well done!");
   score++;
  } else {
   console.log(`Wrong! The correct answer is: ${correctAnswer}`);
  }
}
```

```
 console.log(`Quiz complete! Your final score is: ${score} out of ${maxQuestions}`);
}

// Start the Math Quiz Game
mathQuizGame();
```

193. Color Guessing Game

This program simulates a text-based Color Guessing game. The player is given RGB values, and they need to guess the correct color.

```
function colorGuessingGame() {
 const colors = ["red", "green", "blue", "yellow", "orange", "purple"];
 const maxAttempts = 3;

 console.log("Welcome to the Color Guessing Game!");
 console.log("Guess the color based on the RGB values.");

 for (let attempts = 1; attempts <= maxAttempts; attempts++) {
const randomColor = colors[Math.floor(Math.random() * colors.length)];
const rgbValues = generateRandomRGBValues();

console.log(`RGB: ${rgbValues.join(", ")}`);

const userGuess = prompt("Enter your color guess:").toLowerCase();

if (!colors.includes(userGuess)) {
 console.log("Invalid guess. Please enter a valid color.");
 attempts--;
 continue;
}

if (userGuess === randomColor) {
```

```
    console.log("Congratulations! You guessed the correct color!");
    return;
   } else {
    console.log(`Wrong guess. Remaining attempts: ${maxAttempts - attempts}`);
   }
  }

  console.log("Sorry, you've run out of attempts. The correct color was revealed above.");
}

function generateRandomRGBValues() {
  return Array.from({ length: 3 }, () => Math.floor(Math.random() * 256));
}

// Start the Color Guessing Game
colorGuessingGame();
```

194. Word Scramble Game

This program simulates a text-based Word Scramble game. The player is presented with a scrambled word and needs to unscramble it.

```
function wordScrambleGame() {
 const words = ["javascript", "programming", "web", "hangman", "developer"];
 const maxAttempts = 3;

 console.log("Welcome to the Word Scramble Game!");
 console.log("Unscramble the word and enter your guess.");

 for (let attempts = 1; attempts <= maxAttempts; attempts++) {
const randomWord = words[Math.floor(Math.random() * words.length)];
const scrambledWord = scrambleWord(randomWord);

console.log(`Scrambled word: ${scrambledWord}`);

const userGuess = prompt("Enter your unscrambled word guess:").toLowerCase();

if (!words.includes(userGuess)) {
 console.log("Invalid guess. Please enter a valid word.");
 attempts--;
 continue;
}

if (userGuess === randomWord) {
```

```
    console.log("Congratulations! You unscrambled the word!");
    return;
   } else {
    console.log(`Wrong guess. Remaining attempts: ${maxAttempts - attempts}`);
   }
  }

  console.log("Sorry, you've run out of attempts. The correct word was revealed above.");
}

function scrambleWord(word) {
  return word.split("").sort(() => Math.random() - 0.5).join("");
}

// Start the Word Scramble Game
wordScrambleGame();
```

195. Country Capital Quiz Game

This program simulates a text-based Country Capital Quiz game. The player is asked to match countries with their respective capitals.

```
function countryCapitalQuizGame() {
 const countryCapitalPairs = [
  { country: "France", capital: "Paris" },
  { country: "Japan", capital: "Tokyo" },
  { country: "Brazil", capital: "Brasília" },
  { country: "India", capital: "New Delhi" },
  { country: "Australia", capital: "Canberra" }
];

let score = 0;

console.log("Welcome to the Country Capital Quiz Game!");
console.log("Match the country with its capital:");

for (const { country, capital } of countryCapitalPairs) {
 console.log(`\nCountry: ${country}`);

 const userCapital = prompt(`Enter the capital of ${country}:`).trim();

 if (userCapital.toLowerCase() === capital.toLowerCase()) {
  console.log("Correct! Well done!");
```

```
    score++;
   } else {
    console.log(`Wrong! The correct capital is: ${capital}`);
   }
  }

  console.log(`Quiz complete! Your final score is: ${score} out of ${countryCapitalPairs.length}`);
}

// Start the Country Capital Quiz Game
countryCapitalQuizGame();
```

196. Guess the Animal

This program simulates a text-based Guess the Animal game. The player is provided with clues and needs to guess the correct animal.

```
function guessTheAnimalGame() {
 const animals = [
  { name: "elephant", clues: ["I have a long trunk.", "I am the largest land animal."] },
  { name: "giraffe", clues: ["I have a long neck.", "I am the tallest living terrestrial animal."] },
  { name: "penguin", clues: ["I cannot fly.", "I live in cold climates like Antarctica."] }
 ];

 let score = 0;

console.log("Welcome to the Guess the Animal Game!");
console.log("Read the clues and guess the correct animal:");

for (const { name, clues } of animals) {
 console.log(`\nAnimal Clues:`);
 for (const clue of clues) {
  console.log(clue);
 }

 const userGuess = prompt("Enter your guess:").toLowerCase();

 if (userGuess === name) {
```

```
    console.log("Correct! Well done!");
    score++;
   } else {
    console.log(`Wrong! The correct answer is: ${name}`);
   }
  }

  console.log(`Game complete! Your final score is: ${score} out of ${animals.length}`);
}
// Start the Guess the Animal Game
guessTheAnimalGame();
```

197. Joke Teller

This program simulates a text-based Joke Teller. It randomly selects a joke category and tells a joke from that category.

```
function jokeTellerProgram() {
  const jokeCategories = ["knock-knock", "dad", "animal", "puns"];
  const selectedCategory = jokeCategories[Math.floor(Math.random() * jokeCategories.length)];

  console.log("Welcome to the Joke Teller Program!");
  console.log(`Category: ${selectedCategory.toUpperCase()}`);
  console.log("Get ready for a good laugh!");

  switch (selectedCategory) {
```

```
case "knock-knock":
 console.log("Knock, knock.");
 prompt("Who's there?");
 console.log("Tank.");
 prompt("Tank who?");
 console.log("You're welcome!");
 break;

case "dad":
 console.log("Why did the scarecrow win an award?");
 prompt("I don't know, why?");
 console.log("Because he was outstanding in his field!");
 break;

case "animal":
 console.log("Why don't scientists trust atoms?");
 prompt("I'm not sure. Why?");
 console.log("Because they make up everything!");
 break;

case "puns":
 console.log("I used to be a baker because I kneaded dough.");
 break;

default:
 console.log("Oops! Something went wrong. No joke for you!");
```

```
  }
}

// Start the Joke Teller Program
jokeTellerProgram();
```

198. Shopping List

This program simulates a text-based Shopping List. The user can add items, remove items, and view the current shopping list.

```
function shoppingListProgram() {
 const shoppingList = [];

 console.log("Welcome to the Shopping List Program!");
 console.log("You can add items, remove items, and view your current shopping list.");

 while (true) {
  const action = prompt("Enter 'add' to add an item, 'remove' to remove an item, or 'view' to see your shopping list. Type 'exit' to end the program:").toLowerCase();

  if (action === "exit") {
   console.log("Exiting the Shopping List Program. Goodbye!");
   break;
  }

  switch (action) {
   case "add":
    const newItem = prompt("Enter the item you want to add:");
    shoppingList.push(newItem);
    console.log(`${newItem} added to your shopping list.`);
    break;
```

```
case "remove":
 if (shoppingList.length === 0) {
  console.log("Your shopping list is empty. Nothing to remove.");
 } else {
  const itemToRemove = prompt("Enter the item you want to remove:");
  const indexToRemove = shoppingList.indexOf(itemToRemove);

  if (indexToRemove !== -1) {
   shoppingList.splice(indexToRemove, 1);
   console.log(`${itemToRemove} removed from your shopping list.`);
  } else {
   console.log(`${itemToRemove} not found in your shopping list.`);
  }
 }
 break;

case "view":
 if (shoppingList.length === 0) {
  console.log("Your shopping list is empty.");
 } else {
  console.log("Your Shopping List:");
  for (const item of shoppingList) {
   console.log(`- ${item}`);
  }
```

```
      }
      break;

    default:
      console.log("Invalid action. Please enter 'add', 'remove', 'view', or 'exit'.");
  }
 }
}

// Start the Shopping List Program
shoppingListProgram();
```

199. FizzBuzz Game

The program prints numbers from 1 to 100, but for multiples of 3, it prints "Fizz," for multiples of 5, it prints "Buzz," and for multiples of both 3 and 5, it prints "FizzBuzz":

```
function fizzBuzzGame() {
 console.log("Welcome to the FizzBuzz Game!");
 console.log("Let's play FizzBuzz!");

 for (let i = 1; i <= 100; i++) {
  let output = "";

  if (i % 3 === 0) {
   output += "Fizz";
  }

  if (i % 5 === 0) {
   output += "Buzz";
  }

  console.log(output || i);
 }
}

// Start the FizzBuzz Game
fizzBuzzGame();
```

200. Swap Pairs of Adjacent Digits

This program takes an even-length integer and swaps pairs of adjacent digits. For example, for the number 123456, the program will swap pairs of adjacent digits to produce the number 214365.

```
function swapPairsOfAdjacentDigits(number) {
 // Convert the number to a string to manipulate individual digits
 const numberStr = String(number);
 const length = numberStr.length;

 // Check if the number has an even length
 if (length % 2 !== 0) {
  console.log("Please enter an integer with an even length.");
  return;
 }

 // Swap pairs of adjacent digits
 let swappedNumber = "";
 for (let i = 0; i < length; i += 2) {
  swappedNumber += numberStr[i + 1] + numberStr[i];
 }

 // Convert the result back to an integer
 const result = parseInt(swappedNumber, 10);
 console.log(`Original Number: ${number}`);
```

```
  console.log(`Number with Swapped Pairs of Adjacent Digits: ${result}`);
}

// Example: Swap pairs of adjacent digits of the number 123456
swapPairsOfAdjacentDigits(123456);
```

201. Capitalization Changer

This program takes a string as input and changes the capitalization of all letters. Uppercase letters are converted to lowercase, and lowercase letters are converted to uppercase.

```
function changeCapitalization(inputString) {
 let resultString = "";

 for (const char of inputString) {
  // Check if the character is uppercase
  if (char === char.toUpperCase()) {
   // Convert uppercase to lowercase
   resultString += char.toLowerCase();
  } else {
   // Convert lowercase to uppercase
   resultString += char.toUpperCase();
  }
 }

 console.log(`Original String: ${inputString}`);
 console.log(`String with Changed Capitalization: ${resultString}`);
}

// Example: Change the capitalization of the string "Hello World"
changeCapitalization("Hello World");
```

202. Array Halves Swapper

This program takes an array of integers with an even length and swaps its two halves.

```
function swapArrayHalves(arr) {
 const length = arr.length;

 // Check if the array has an even length
 if (length % 2 !== 0) {
  console.log("Please provide an array with an even length.");
  return;
 }

 // Calculate the midpoint of the array
 const midpoint = length / 2;

 // Swap the two halves of the array
 for (let i = 0; i < midpoint; i++) {
  const temp = arr[i];
  arr[i] = arr[midpoint + i];
  arr[midpoint + i] = temp;
 }

 console.log("Original Array: " + arr.join(", "));
 console.log("Array with Swapped Halves: " + arr.join(", "));
}
```

```
// Example: Swap halves of the array [1, 2, 3, 4, 5, 6]
swapArrayHalves([1, 2, 3, 4, 5, 6]);
```

203. Sum of Digits in String

This program computes the sum of all the digits that occur in a given string. It identifies digits in the string and adds them up to calculate the total sum.

```
function sumOfDigitsInString(inputString) {
 let digitSum = 0;

 for (const char of inputString) {
  // Check if the character is a digit
  if (/\d/.test(char)) {
   // Convert the digit character to its numerical value and add to sum
   digitSum += parseInt(char, 10);
  }
 }

 console.log(`Original String: ${inputString}`);
 console.log(`Sum of Digits in the String: ${digitSum}`);
}

// Example: Compute the sum of digits in the string "abc123xyz456"
sumOfDigitsInString("abc123xyz456");
```

204. Sum of Cubes

This program computes the sum of cubes of all integers from 1 to a given integer. It calculates the cube of each integer in the specified range and adds them up to find the total sum.

```
function sumOfCubes(upToInteger) {
 let cubesSum = 0;

 for (let i = 1; i <= upToInteger; i++) {
  // Calculate the cube of each integer and add to sum
  cubesSum += Math.pow(i, 3);
 }

 console.log(`Sum of Cubes from 1 to ${upToInteger}: ${cubesSum}`);
}

// Example: Compute the sum of cubes up to the integer 5
sumOfCubes(5);
```

205. Maximum Integer for Sum

This program finds the maximum integer `n` such that the sum of integers from 1 to `n` (inclusive) does not exceed a given target sum `targetSum`. It calculates the sum using the formula `Sum = n * (n + 1) / 2`.

```
function findMaxIntegerForSum(targetSum) {
 let currentSum = 0;
 let maxInteger = 0;

 while (currentSum + maxInteger + 1 <= targetSum) {
  maxInteger++;
  currentSum += maxInteger;
 }

 console.log(`Maximum Integer (n) for Sum <= ${targetSum}: ${maxInteger}`);
}

// Example: Find the maximum integer for the sum <= 15
findMaxIntegerForSum(15);
```

206. URL Breakdown

This program breaks a URL address into its parts and puts them into an array. The URL structure considered is `scheme://domain[/:path]`, where `path` is optional.

```
function breakURL(url) {
 const urlParts = {};
 const urlRegex = /^(\w+):\/\/([\w.-]+)(\/.*)?$/;
 const matches = url.match(urlRegex);

 if (!matches) {
  console.log("Invalid URL format.");
  return;
 }

 urlParts.scheme = matches[1];
 urlParts.domain = matches[2];
 urlParts.path = matches[3] || "";

 console.log("URL Parts:");
 console.log(urlParts);
}

// Example: Breakdown the URL "https://www.example.org/page"
breakURL("https://www.example.org/page");
```

207. Sort Strings by Length

This program sorts the strings of a given array of strings in order of increasing length. It uses the `Array.sort` method to achieve the desired order.

```
function sortStringsByLength(stringsArray) {
  const sortedArray = stringsArray.slice().sort((a, b) => a.length - b.length);

  console.log("Original Array of Strings:");
  console.log(stringsArray);
  console.log("Array of Strings Sorted by Length:");
  console.log(sortedArray);
}

// Example: Sort strings by length
sortStringsByLength(["apple", "banana", "orange", "kiwi", "grape"]);
```

208. Simplify Absolute Path

This program simplifies a given absolute path for a file in Unix-style. It takes a string representing the absolute path and simplifies it by resolving any redundant slashes (`//`), parent directory references (`..`), and current directory references (`.`).

```
function simplifyAbsolutePath(path) {
 const parts = path.split('/');
 const simplifiedParts = [];

 for (const part of parts) {
  if (part === '..') {
   simplifiedParts.pop(); // Move up one level for '..'
  } else if (part !== '' && part !== '.') {
   simplifiedParts.push(part);
  }
 }

 const simplifiedPath = '/' + simplifiedParts.join('/');

 console.log(`Original Absolute Path: ${path}`);
 console.log(`Simplified Absolute Path: ${simplifiedPath}`);
}

// Example: Simplify an absolute path
simplifyAbsolutePath('/home/user/../documents/./file.txt');
```

209. Count Common Elements in Arrays

This program finds the number of common elements between two arrays. It compares the elements in both arrays and counts the occurrences of common elements.

```
function countCommonElements(arr1, arr2) {
 const commonElements = arr1.filter(element => arr2.includes(element));
 const numberOfCommonElements = commonElements.length;

 console.log("Array 1:", arr1);
 console.log("Array 2:", arr2);
 console.log("Common Elements:", commonElements);
 console.log("Number of Common Elements:", numberOfCommonElements);
}

// Example: Count common elements between two arrays
countCommonElements([1, 2, 3, 4, 5], [3, 4, 5, 6, 7]);
```

210. Check Same Digits in a Number

This program checks whether all the digits in a given number are the same or not. It converts the number to a string and compares each digit to the first digit.

```
function areAllDigitsSame(number) {
 const numberStr = String(number);
 const firstDigit = numberStr[0];

 for (let i = 1; i < numberStr.length; i++) {
  if (numberStr[i] !== firstDigit) {
   console.log(`Digits in ${number} are not all the same.`);
   return;
  }
 }

 console.log(`Digits in ${number} are all the same.`);
}

// Example: Check if all digits in the number 22222 are the same
areAllDigitsSame(22222);
```

211. Rightmost Round Number Position

This program finds the position of the rightmost round number in an array of integers. A round number is informally considered to be an integer that ends with one or more zeros. The program iterates through the array from right to left and identifies the rightmost round number, if any.

```
function rightmostRoundNumberPosition(arr) {
 for (let i = arr.length - 1; i >= 0; i--) {
  if (arr[i] % 10 === 0) {
   console.log(`Rightmost Round Number: ${arr[i]}, Position: ${i + 1}`);
   return;
  }
 }

 console.log("No round numbers found in the array. Position: 0");
}

// Example: Find the position of the rightmost round number
rightmostRoundNumberPosition([123, 450, 678, 900]);
```

212. Reverse Bits of 16-Bit Unsigned Short Integer

This program reverses the bits of a given 16-bit unsigned short integer. It converts the integer to its binary representation, reverses the bits, and converts it back to an integer.

```
function reverseBits16BitUnsignedShort(integer) {
  const binaryRepresentation = integer.toString(2).padStart(16, '0');
  const reversedBinary = binaryRepresentation.split('').reverse().join('');
  const reversedInteger = parseInt(reversedBinary, 2);

  console.log(`Original Integer: ${integer}`);
  console.log(`Binary Representation: ${binaryRepresentation}`);
  console.log(`Reversed Binary: ${reversedBinary}`);
  console.log(`Reversed Integer: ${reversedInteger}`);
}

// Example: Reverse bits of the 16-bit unsigned short integer 5678
reverseBits16BitUnsignedShort(5678);
```

213. Greater Than 15 Checker

This program tests whether a given integer is greater than 15. If the integer is greater than 15, it returns the given number; otherwise, it returns 15.

```
function greaterThan15Checker(number) {
 const result = number > 15 ? number : 15;
 console.log(`Given Number: ${number}`);
 console.log(`Result: ${result}`);
}

// Example: Test whether the number 20 is greater than 15
greaterThan15Checker(20);
```

214. Replace First Digit with $

This program replaces the first digit in a string (assuming the string has at least one digit) with the `$` character.

```
function replaceFirstDigitWithDollar(inputString) {
 const modifiedString = inputString.replace(/\d/, '$');

 console.log(`Original String: ${inputString}`);
 console.log(`Modified String: ${modifiedString}`);
}

// Example: Replace the first digit with $ in the string "abc123xyz456"
replaceFirstDigitWithDollar("abc123xyz456");
```

215. Prefix Sums

This program creates an array of prefix sums for a given array. The prefix sum of a sequence is a new sequence where each element is the sum of all preceding elements in the original sequence.

```
function prefixSums(inputArray) {
 let prefixSum = 0;
 const prefixSumArray = [];

 for (const num of inputArray) {
  prefixSum += num;
  prefixSumArray.push(prefixSum);
 }

 console.log("Original Array:", inputArray);
 console.log("Prefix Sums Array:", prefixSumArray);
}

// Example: Create prefix sums for the array [1, 2, 3, 4, 5]
prefixSums([1, 2, 3, 4, 5]);
```

216. Next Prime Number

This program finds the smallest prime number strictly greater than a given number. It checks numbers one by one starting from the given number until it finds the next prime.

```
function isPrime(num) {
 if (num < 2) return false;
 for (let i = 2; i <= Math.sqrt(num); i++) {
  if (num % i === 0) return false;
 }
 return true;
}

function nextPrimeNumber(givenNumber) {
 let nextNumber = givenNumber + 1;

 while (!isPrime(nextNumber)) {
  nextNumber++;
 }

 console.log(`Given Number: ${givenNumber}`);
 console.log(`Next Prime Number: ${nextNumber}`);
}

// Example: Find the next prime number greater than 10
nextPrimeNumber(10);
```

217. Reverse Order of Bits

This program reverses the order of bits in an integer. It converts the integer to its binary representation, reverses the bits, and converts it back to an integer.

```
function reverseOrderOfBits(integer) {
 const binaryRepresentation = integer.toString(2).padStart(8, '0'); // Assuming 8 bits for simplicity
 const reversedBinary = binaryRepresentation.split('').reverse().join('');
 const reversedInteger = parseInt(reversedBinary, 2);

 console.log(`Original Integer: ${integer}`);
 console.log(`Binary Representation: ${binaryRepresentation}`);
 console.log(`Reversed Binary: ${reversedBinary}`);
 console.log(`Reversed Integer: ${reversedInteger}`);
}

// Examples: Reverse order of bits for given integers
reverseOrderOfBits(14);
reverseOrderOfBits(56);
reverseOrderOfBits(234);
```

218. Pyramid

This program generates a triangle pattern using ASCII characters and displays it in the console.

```
function generateAsciiTriangle(height) {
 for (let i = 1; i <= height; i++) {
  const spaces = ' '.repeat(height - i);
  const stars = '*'.repeat(i * 2 - 1);
  console.log(spaces + stars);
 }
}

const triangleHeight = 5;
generateAsciiTriangle(triangleHeight);
```

In every line of code, they have woven a story of innovation and creativity. This book has been your compass in the vast world of JavaScript.

Close this chapter knowing that every challenge overcome is an achievement, and every solution is a step toward mastery.

Your code is the melody that gives life to projects. May they continue creating and programming with passion!

Thank you for allowing me to be part of your journey.

With gratitude,

Hernando Abella

500+ JavaScript Programs

Discover Other Useful Resources at:

www.hernandoabella.com